UNIVERSITY OF TWENTE.
EEMCS faculty
Nanoelectronics group
BRAINS

An Oscillatory Neural-Network architecture based on Dopant Network Processing Units

Bachelor Thesis

Marijn Buitenweg

Supervisors:
prof.dr.ir. Wilfred G. van der Wiel
Mohamadreza Zolfagharinejad MSc
prof.dr. Christoph Brune

version 1

Enschede, July 2023

Abstract

The amount of data is growing faster than ever, yet the crucial task of comprehending and deriving meaning from this information persists. This seemingly insatiable demand for inference is to be met within an obviously bounded energy budget, that must be made to stretch as far as possible. Consequently, alternative computing paradigms are being investigated with the aspiration of surpassing current energy-efficiency standards and achieving higher performance. Among these paradigms are the Dopant Network Processing Units (DNPUs), a class of high-dimensionally tunable non-linear silicon-based devices that have recently been shown to be capable of energy- and footprint-efficient compute. Oscillatory Neural Networks (ONNs) form a less recent computing paradigm aiming at raising efficiency by exploiting the synchronization phenomena found in oscillator networks to compute in phase and frequency instead of amplitude. In this BSc Assignment, we investigate ways of utilizing DNPUs to create an enhanced ONN architecture. A phase-computing DNPU-based ONN architecture is described. Then, an ONN simulator is developed and used to demonstrate that the described architecture can perform associative memory and classification tasks. It is shown that these tasks can also be performed when the coupling between oscillators is nonlinear, especially if the network complexity is low. The set of tasks shown in this work leave the computational capabilities of the DNPUs in the ONN underutilized, leaving room for further investigation.

Glossary

DNPU	Dopant Network Processing Unit
ONN	Oscillatory Neural Network
HNN	Hopfield Neural Network
MHNN	Modern HNN
DAM	Dense Associative Memory
PRC	Phase Response Curve
NDR	Negative Differential Resistance
SNR	Signal to Noise Ratio
AAM	Auto-Associative Memory
HAM	Hetero-Associative Memory

Contents

Chapter 1

Introduction

As today's data processing requirements grow, an increasing amount of inspiration is being drawn from the most efficient processor available thus far: the brain. This neuromorphic inspiration has led to the development of many different computing paradigms, many of which are aimed at approaching the brain's energy efficiency. Oscillatory neural networks (ONNs) take inspiration from oscillatory phenomena found in the brain to perform computations using oscillator networks. Traditionally, analog computing is performed using signal amplitude, but this inexorably links the computation's precision to the analog computer's energy usage via its Signal to Noise Ratio (SNR). By computing in phase and frequency, ONNs can mitigate this problem if they are implemented well. ONNs are not without problems. They scale poorly and require the connections between oscillators to be programmable (see section 2.1).
Dopant Network Processing Units (DNPUs) are nanoelectronic devices that leverage hopping conduction between dopant atoms to produce highly tuneable input (voltage)-output (current/-voltage) curves. These devices are capable of efficient compute for classification and other tasks (See section 2.2). Their substantial amount of configurability makes them a prime candidate to alleviate the problems found in today's ONN architectures, either by increasing their capabilities or by decreasing the complexity of their hardware.

These problems lead to the following research questions:

- What could a DNPU-based ONN architecture look like?
- What could this architecture be used for?

To answer these questions this report will first provide an overview of some of the required theory, then a DNPU-based ONN architecture will be described. This ONN architecture will be simulated according to the methodology outlined in chapter 4. The simulated architecture will then be used to perform some tasks in chapter 5.

Chapter 2

Theory

This chapter covers the preliminary knowledge that is required to understand the work done for this assignment. First, ONNs will be explained. Then, a brief explanation of the DNPU is given.

2.1 Oscillatory Neural Networks

ONNs are networks of weakly connected oscillators. This means that the oscillators in an ONN are connected, and will affect each other, but those connections do not significantly influence the magnitude of the oscillator's signals. Instead, these connections affect the temporal characteristics of the produced signals, altering their frequencies and phases [1]. The constraints on what can be used as an oscillator in a system like this are very loose. The oscillator can produce any kind of signal, as long as it is periodic, and the oscillator can be weakly coupled.

2.1.1 Synchronization

In ONNs, information is carried by the relative phases of the oscillators, and processing is done using the synchronization dynamics of the system. The synchronization dynamics of a non-linear system are non-trivial to derive and use, therefore it is common to approximate a system using only the dominant harmonics of the connection functions [1]. This approximation forms the basis of the Kuramoto model for synchronization phenomena. The Kuramoto model describes the dynamics of a fully interconnected ONN with N oscillators with coupling strengths K_{ij} with:

$$\dot{\theta}_i = \omega_i + \sum_{j=1}^{N} K_{ij} sin(\theta_j - \theta_i), \tag{2.1}$$

wherein ω is the frequency of an oscillator, and θ is the phase of an oscillator [3]. This model is somewhat simplistic, but its implications are important to understand when working with ONNs. From equation 2.1 follows that if the differences in oscillator frequencies ω_i are small, and the phase differences between oscillators i and j are not exactly $0°$ or $180°$, a positive coupling strength between them will cause them to synchronize to a phase difference of $0°$. While a negative coupling strength will cause them to synchronize to a phase of $180°$ in this situation. It is this basic principle that is most commonly applied to make ONNs compute in phase, as it effectively turns an oscillator into a threshold unit. Examples of ONNs using this principle can be found in [4][5][6][7]. The process of mapping the coupling coefficient matrix K_{ij} to hardware parameters depends on how, and what type of, oscillators are coupled.

2.1.2 Coupling techniques

Many different ONN architectures have been developed, using many different methods to couple oscillators. We can categorize these architectures into two main categories:

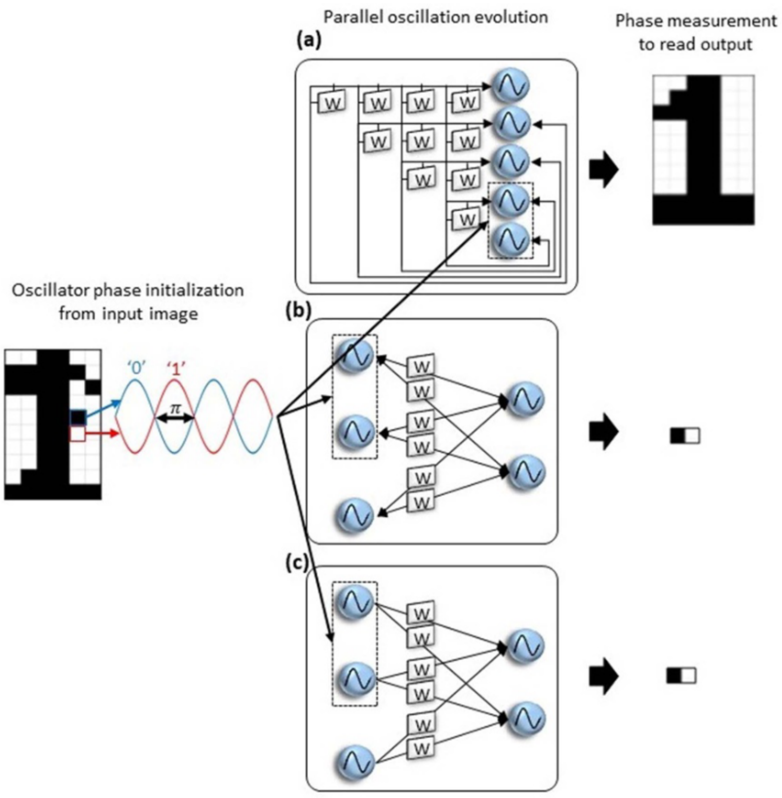
Parallel oscillation evolution
Phase measurement to read output
(a)
W
Oscillator phase initialization from input image
(b)
'0'
'1'
π
(c)

Figure 2.1: ONN architectures for (a) fully connected auto-associative memory, (b) bidirectional two-layer hetero-associative memory, and (c) feedforward two-layer classification. Figure is from [2].

1. **Direct current injection:** Requires a coupling element for every connection. A coupling element injects a current into one or both of the oscillators it couples. The most simple example of this is a resistor connecting the outputs of two oscillators. Some types of coupling elements are not programmable, and bigger networks require more coupling elements per oscillator. [4][8][9].

2. **Frequency multiplexing:** Runs sinusoidal oscillators at different frequencies and connects a number of them to a common medium. The required dynamic coupling behaviour is then introduced by applying a carefully chosen quasi-periodic signal to the common medium. If this signal contains the appropriate frequency components, the system will behave according to the principles described earlier [5][10]. Larger networks require more frequency precision.[6][10].

2.1.3 Connection to Hopfield Neural Networks

Before elaborating on the details of ONNs, it is important to grasp the basics of the computational model that forms the theoretical basis for most ONN workloads: the Hopfield neural networks (HNNs). This model, popularised in 1982 by J.J. Hopfield [11], is used to describe associative memories. An associative memory stores a number of patterns, and when given input will output the pattern that is most closely associated with it. Broadly speaking, there are two ways of using associative memory:

1. **Auto-Associative Memory (AAM):** Output has the same dimensions and meaning as the input. When given an input, the output will be the pattern that is most similar to the

input. This type of system is also known as content-addressable memory.

2. **Hetero-Associative Memory (HAM):** Output has different dimensions than the input. This is used to associate one piece of data with another. This mode of operation is used for tasks like classification.

An HNN is a single-layer fully interconnected Recurrent Neural Network (RNN), where the nodes consist of binary threshold units (so $output = sgn(\sum input)$). The behaviours of many ONNs are close approximations of this type of network, to the point that most ONN architectures are tested for their associative memory capabilities.
To use an associative memory, one needs to be able to teach it what patterns to store. This is done by applying a learning rule to the set of patterns to be stored. This yields a matrix of weights that can be directly used in an HNN, but can also be used in an ONN if the correct hardware-dependent transformations are applied. Defining ξ_i^μ as the i^{th} element of pattern μ, w_{ij}^μ as the matrix of weights after learning pattern μ, and N as the number of nodes in the network, the Hebbian learning rule is as follows:

$$w_{ij}^0 = 0 \text{ and } w_{ij}^\mu = w^{\mu-1} + \frac{1}{N}\xi_i^\mu \xi_j^\mu. \tag{2.2}$$

This learning rule is trivial to implement but sub-optimal. In 1999, A. Storkey formulated an improved learning rule:

$$w_{ij}^0 = 0 \text{ and } w_{ij}^\mu = w^{\mu-1} + \frac{1}{N}\xi_i^\mu \xi_j^\mu - \frac{1}{N}\xi_i^\mu h_{ji}^\mu - \frac{1}{N}\xi_i^\mu h_{ji}^\mu - \frac{1}{N}\xi_j^\mu h_{ij}^\mu, \text{ where} \tag{2.3}$$

$$h_{ij}^\mu = \sum_{k=1,k\neq i,j}^{N} w_{ik}^{\mu-1}\xi_k^\mu \tag{2.4}$$

This learning rule significantly improves on the storage capacity (the number of patterns that can be retrieved with low error rates) that could be achieved using the Hebbian learning rule [12]. The storage capacities that are normally achieved using the Hebbian learning rule are around $0.14N$. The Storkey learning rule outperforms this significantly but still scales linearly with N.
In more recent years, starting with the 2016 paper from Krotov and Hopfield [13], denser versions of HNNs have been developed. These exploit additional non-linearity to make the memory capacity of HNNs scale super-linearly, or even exponentially [14]. These newer HNNs are known as Modern HNNs (MHNNs) or Dense Associative Memories (DAMs). MHNNs have been generalized to work with continuous states, at which point their functionality is interchangeable with that of the attention mechanism in transformers [15]. These newer developments do not appear to have been carried over to the field of ONNs.

2.1.4 ONN scaling

One of the biggest problems with ONNs is that their capabilities (memory capacity) grow linearly, while the hardware becomes quadratically harder to realize. For ONNs that require coupling elements, the number of coupling elements scales according to N^2. For frequency multiplexed ONNs, the required frequency precision increases with N^2 (for the system in [6]) or $N^{1.58}$ (for the system in [10]). To improve this situation, it would be good if the architecture designed in this assignment could either alleviate these super-linear hardware requirements or improve the memory capacity scaling.
The maximum size of Current injection-based ONNs is also limited by the quantity of disturbing signals each coupling element produces. How strict this limit is depends on several factors, among which are the nature of the disturbance, and the Signal to Noise Ratio (SNR) of the oscillator unit [4].
There have also been investigations into using different network topologies to cut down the hardware complexity. For HAM tasks, feedforward topologies can be used instead of the traditional fully connected one [2]. This significantly reduces the number of connections, replacing the N^2 scaling with NK, where K is the size of the output.

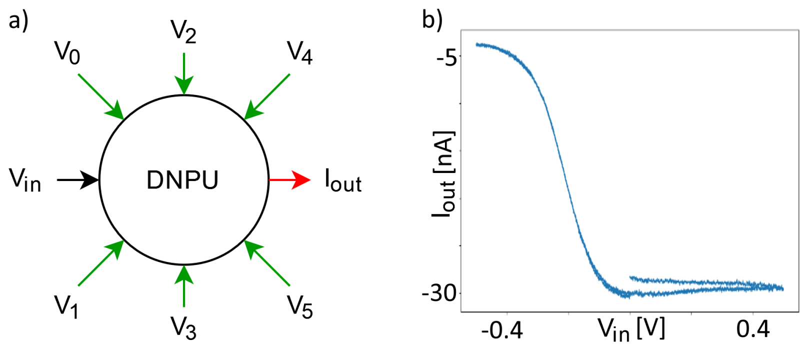

Figure 2.2: Diagram (a) of the DNPU with one input electrode V_{in}, six control electrodes V_{0-5}, and one output electrode I_{out}. Graph (b) shows the output current I_{out} when V_{in} is swept from -0.5V to 0.5V while control voltages V_{0-5} are kept constant.

2.2 Dopant Network Processing Unit

The DNPU is a nanoelectronic device that has been the subject of research at the NanoElectronics (NE) group at the University of Twente (UT). In this section, the device and its operating principles will be described.
The DNPU was first described in a 2020 paper by Chen et al [16]. In a bulk of doped silicon, a DNPU consists of an area (300 nm of diameter) of silicon with a low doping concentration (using arsenic or boron dopants) surrounded by multiple electrodes. Through the phenomenon of hopping conduction, this device is capable of emulating the behaviour of a simple ANN whose complexity exceeds that of a single perceptron. It has been shown that DNPUs can be used to implement basic logic gates, feature filters for handwritten digit classification, and more[17].
The behaviour of the device can be tuned by changing the voltages applied to its control electrodes. There are multiple ways of finding what control voltages are needed to produce the desired behaviour. It can be done on-chip by using a genetic algorithm, or by using an off-chip surrogate model based on measurements from the device[18].
The device is also capable of exhibiting Negative Differential Resistance (NDR) (as can be seen in [19]). NDR can be used to apply positive feedback to a resonator to build an oscillator. The fact that the DNPUs NDR is tuneable was the initial reason for investigating its utility in ONNs. The aim of this assignment has since moved in the direction of more broadly utilizing the DNPU's tuneability.
The response of a DNPU can exhibit some hysteresis, which may place an upper bound on the operating frequency of the ONN architecture it is used in. The hysteresis has been ignored in this assignment, based on the assumption that the operating frequency of the ONN can be decreased far enough that the effects of the DNPU's hysteresis vanish.

Chapter 3

Design process

To design an ONN that utilizes DNPUs, different types of ONNs were examined to see if some of their problems could be solved by introducing DNPUs. In general, the problem that occurs in most of today's ONN architectures is that their capabilities scale poorly with their complexity. In some of these architectures, this complexity is encountered in the form of a large number of coupling elements. While in others, this complexity manifests itself as increasingly stringent frequency precision requirements. There are several ways one could try to loosen these restrictions, some of which have been investigated in this chapter. First, the main architecture used in this assignment will be described. Then, some alternative ideas that were also considered will be mentioned.

3.1 A DNPU-coupled ONN

As described in section 2.1, phase-computing ONNs that are coupled using differential current injection suffer from the fact that their hardware requirements scale with N^2. Replacing the coupling elements with an array of DNPUs introduces an unprecedented amount of configurability into the system. The idea is that by harnessing this enhanced configurability one can increase the capabilities of the network. From a high-level perspective, this makes sense. Adding more dimensions of configurability should allow one to encode more information into the network. However, it is not obvious how to use these additional degrees of freedom in a useful way. Krotov and Hopfield's work from 2016 [13] may provide some good hints.

3.1.1 Towards an Oscillatory Dense Associative Memory Network

Considering the close relationship between ONNs and HNNs in literature and principle of operation, it makes sense to try to apply the more recent developments in the field of HNNs to an ONN. Since the differences between an old-fashioned HNN and a DAM follow from the change in its energy function, and ONN behaviour can also be described using an energy function[1, p. 285][8] based on the coupling behaviour, it may be possible to construct an Oscillatory Dense Associative Memory Network (ODAMN) by introducing the right kind of non-linearity into the system. The general ONN energy function is:

$$U(\phi_0, ..., \phi_N) = \frac{1}{2}\sum_i \sum_j R_{ij}(\phi_j - \phi_i), \tag{3.1}$$

where

$$R_{ij}(\chi) = \int_0^\chi H_{ij}(t)dt, \tag{3.2}$$

and H_{ij} describes the effect of oscillator j on oscillator i via their coupling as described by

$$\dot{\phi}_i = \sum_j H_{ij}(\phi_j - \phi_i). \tag{3.3}$$

These definitions are from [1], and hold as long as the oscillators in the network have the same frequency and H_{ij} exhibits odd symmetry.
Since H_{ij} describes the change in phase caused by some difference in phase between two oscillators, for differential current injection-based architectures it can also be described using the following equation:

$$H_{ij}(\phi_\Delta) = \Gamma(\phi_i) \cdot I_i^{in}(\phi_\Delta), \tag{3.4}$$

where $\Gamma(\phi_i)$ is the phase-dependent phase sensitivity of the oscillator, and $I_i^{in}(\phi_\Delta)$ is the current injected into the oscillator's hysteresis provider (usually a resonator) based on the phase difference between the two oscillators. This equation was based on work in [8]. In an architecture where DNPUs are used for coupling, $I_i^{in}(\phi_\Delta)$ is defined by the response of the DNPUs. $\Gamma(\phi_i)$ depends on the hysteresis provider that is used to sustain the oscillation.
To make an ODAMN work, the shape of the energy function U would need to look like the following energy function, given that ξ_i^μ represents element i of pattern μ (consisting of values of -1 and 1), there are K patterns, and σ_i represents element i of the state of the network:

$$E = -\sum_{\mu=1}^{K} F(\sum_i \xi_i^\mu \sigma_i), \tag{3.5}$$

where $F(x)$ can be a rectified polynomial (as used in [13]), or e^x (as used in [14]). It is unclear if there is a valid combination of $I_i^{in}(\phi_\Delta)$ and $\Gamma(\phi_i)$ that would approximate E sufficiently well to be able to function like an ODAMN.

3.1.2 The architecture

Since the investigation of the ODAMN remained inconclusive, it also did not add any new requirements the ONN should meet. All that is needed for an ONN to be coupled using DNPUs is a structure that connects an array of DNPUs to different oscillators, sums their output currents, and uses that signal to drive a resonator. The DNPUs should be connected in such a way that their behaviour will be similar to the behaviours seen in earlier research such as in [16]. The circuit displayed in figure 3.1 should meet these requirements. The designed oscillator unit has one DNPU for each other oscillator in the network. Each of these DNPUs is connected to one other oscillator and its own oscillator. The outputs of the DNPUs are summed by an op amp-based current adder. The output of this adder is fed into a resonator. The role of this resonator can be fulfilled by anything that has a high peak in its frequency response. The height of this peak determines the resonator's ability to filter out unwanted frequencies, which is one of the factors determining the upper limit of the number of oscillators that can be connected to form a usable network. In practice, this means that many resonators are suitable for use in this architecture. The network should contain two additional oscillators that can be unidirectionally coupled to the rest of the network: One reference oscillator, and one neutral oscillator that always has a 90° phase shift from the reference. These added oscillators add two extra DNPUs to the DNPU array of every oscillator unit. This defines the following relation between the number of required DNPUs and the network size:

$$N(N-1) + 2N = N(N+1). \tag{3.6}$$

This only applies when the network is fully interconnected, for feedforward configurations the following expression is used:

$$NK + K + N, \tag{3.7}$$

Where NK represents the feedforward connections, the K term is for the connections from the neutral oscillator to the output layer, and the N term is for the connections from the reference oscillator to the input layer. Note that for the feedforward network, the couplings to the reference and neutral oscillators probably do not need to be implemented using DNPUs.

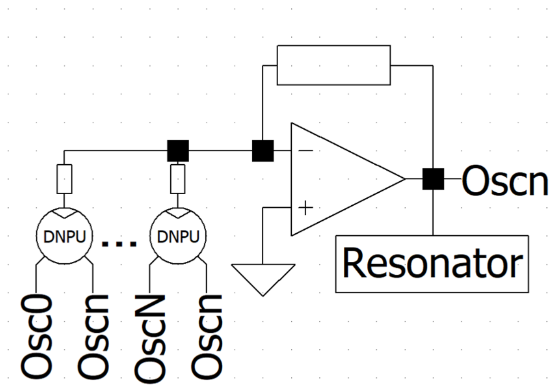

Figure 3.1: Schematic of the oscillator unit n in the designed architecture with N total oscillators. DNPU control electrodes are not depicted.

Chapter 4

Simulation methodology

To validate and analyze the designed architecture, a simulation setup was created, which consists of two main parts:

- A circuit simulator, in this case, LTSpice
- A program controlling the circuit simulator, in this case, a Python script

An overview of how these components fit together can be seen in figure 4.1. These components will be described in detail in sections 4.2 and 4.3. The next section will describe the questions this simulation setup is - and is not - required to answer.

4.1 Requirements

The simulator is supposed to shed light on the computational capabilities of the system being simulated. In practice, this means that the magnitudes of many quantities in the simulation can be arbitrarily chosen as long as the phase behaviour of the system is not affected. Among the quantities that can be arbitrarily chosen are the amplitudes of the signals, and the operating frequency.

4.2 Circuit Simulator

The process of assembling the simulation setup began with some very basic experimentation in a circuit simulator. LTSpice was used for the circuit simulations.

4.2.1 Resonator

The designed architecture does not place many constraints on the resonator that is used to sustain the oscillations. The resonator frequency was arbitrarily chosen to be 159Kz. The Q factor of the resonator was chosen to be from the lower end of what could be realistically expected, resulting in a Q factor of 10.

4.2.2 Oscillator Unit

The array of DNPUs feeding into the current adder (as seen in 3.1) can be simplified into a single arbitrary controlled current source for the purposes of this simulation. This current source is combined with the previously described resonator, and a source of white noise, to form a basic oscillator unit that can be replicated as many times as needed. This results in a network where all the coupling effects influencing an oscillator are encoded in the function that is set to its corresponding controlled current source. These functions must be reformulated for every different

ONN computation that is simulated. Doing this by hand takes a lot of time, and is error-prone. This problem can be avoided by automating this process, which is done using the system described in the next section.

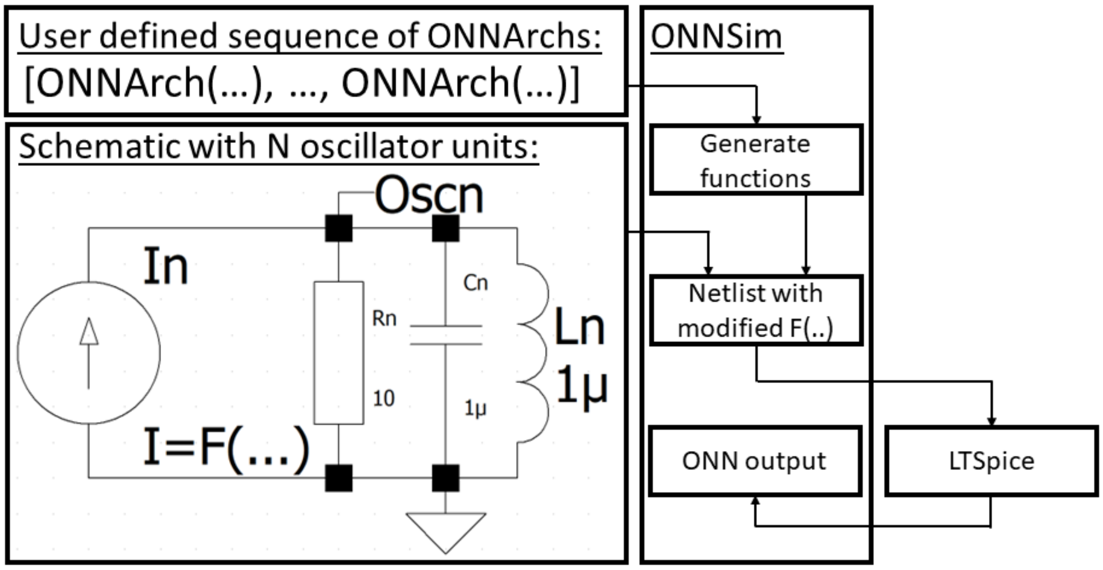
User defined sequence of ONNArchs:
[ONNArch(...), ..., ONNArch(...)]
Schematic with N oscillator units:
Oscn
In
Rn
10
Cn
1μ
Ln
1μ
I=F(...)
ONNSim
Generate functions
Netlist with modified F(..)
ONN output
LTSpice

Figure 4.1: Overview of the simulation setup. The noise source has been left out for readability.

4.3 Simulator control script

The simulator is controlled by a Python script that utilizes the PyLTSpice [20] library to set the formulas in the programmable current sources such that the network is coupled exactly as the user has specified. The script contains two main classes, these being:

- **ONNArch**: Describes an ONN configuration that exists for some amount of time.
- **ONNSim**: Programmes and runs simulations described by a sequence of ONNArchs.

4.3.1 ONNArch

An ONNArch contains all the information needed to program the simulation for some amount of time. It stores a matrix of coupling coefficients. These coefficients can be supplied by the user, or calculated using a selection of algorithms. These include the Hebbian and Storkey learning rules. It also contains a factor that is applied to the coefficients resulting from these learning rules to allow the user to tweak the network to make sure the couplings are weak enough to not cause problematic changes in oscillator amplitude. To allow for experimentation beyond the standard linear coupling functions, the ONNArch contains a matrix defining the type of coupling that corresponds with each entry in the coupling coefficient matrix. The available coupling types are described in table 4.1. The patterns supplied to the learning rule functions can be 1-or-2 dimensional collections of elements that quantize to 0 or 1 when cast to an integer. The ONNArch also keeps track of what oscillators should be coupled to the neutral oscillator.

4.3.2 ONNSim

ONNSims run sequences of ONNArchs. The ONNSim generates all the current source functions and coordinates the running of simulations. The functions that are generated for each ONNArch

Name	Description
Uncoupled	0
Linear	$c \cdot x$
Polynomial	$c \cdot x^k$
Exponential	$c \cdot e^x$
Symmetric Exponential	$c \cdot sgn(x) \cdot (e^{\|x\|} - 1)$
NDR	Changes NDR by $c \cdot x$

Table 4.1: Different coupling types supported by the ONNArch. Where $x = V_i - V_{self}$, and c is the coupling coefficient. k is the order of the polynomial coupling function.

are multiplied by $u(t-t_{start})-u(t-t_{end})$, where $u(t)$ is the unit step function, t_{start} is the starting time of the ONNArch, and t_{end} is the end time of the ONNArch. This is done to make sure that only the correct ONNArch is active at any given time.

Output extraction

To extract the output phases of the simulation, the ONNSim takes all the waveforms of the oscillators that were part of the final ONNArch, and then determines their phase-coded value according to the following logic:

$$\int_{T(1-p)}^{T} v_0(t)v_i(t)dt > 0 \Rightarrow 1, \text{ otherwise } 0 \tag{4.1}$$

where T is the total run time of the simulation, and p is the fraction of the simulation time in which the system is in its output state (usually $p \leq 0.1$). Assuming that the oscillator amplitudes are similar, this determines and quantizes the phase of each non-reference oscillator to be either 0 or 180 degrees, and encodes this phase into a 1 (for 0°) or 0 (for 180°). This binary sequence can then be extracted from the ONNSim as an array, string, or 2-dimensional array.

Chapter 5

Results

To ascertain the functionality of the architecture described in chapter 3, the simulation setup is utilized to make the architecture perform standard associative memory tasks.

5.1 Auto-associative memory

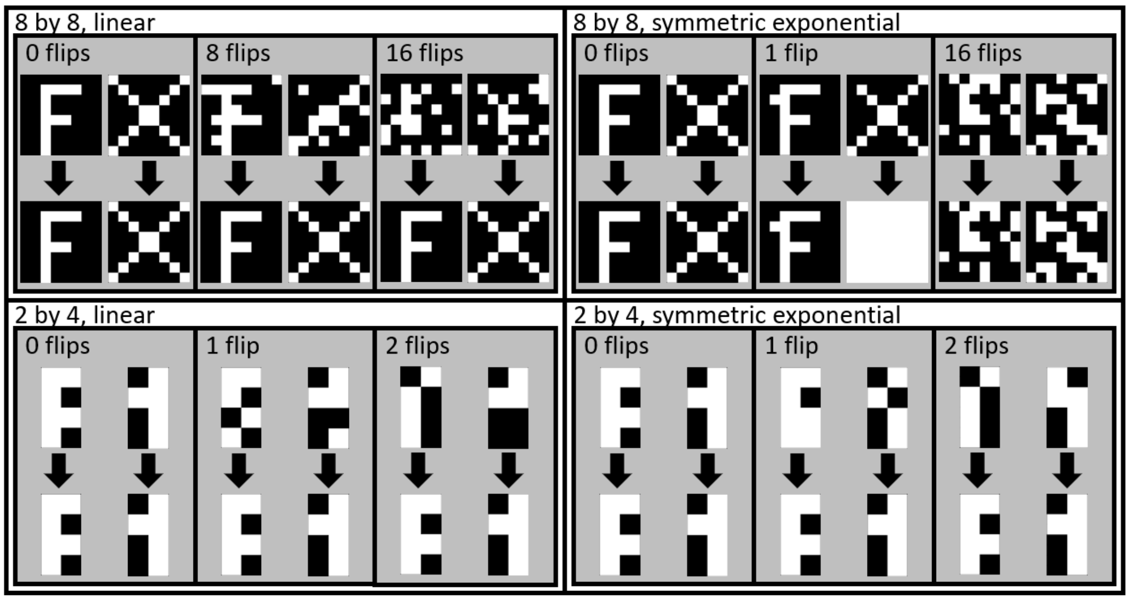

Figure 5.1: Results of the associative memory task performed with linear and symmetric exponential coupling functions. The number of 'flips' indicates the hamming distance between the taught pattern and the input.

A standard way of demonstrating the functionalities of an ONN is to make it perform an associative memory task. To make the simulated system perform this task, two patterns were taught to it by means of the Storkey learning rule. Modified versions of these patterns were then presented to it as input. If the system works, it should then converge to the pattern that most closely resembles the presented input. This task was performed on ONNs of two different sizes: an 8-oscillator ONN, and a 64-oscillator ONN. On each of these, the task was performed for each of the following coupling types: linear, exponential, symmetric exponential, and 3^{rd}-order polynomial (see table 4.1 for coupling type definitions).
According to equation 3.6, the 8-oscillator ONN would require 72 DNPUs to be implemented in

hardware. The 64-oscillator system would require 4160 DNPUs, which is more than could be reasonably justified for a system performing a task like this. Despite this, testing the 64-oscillator ONN provides a good stress test.
The taught patterns were modified by flipping a predetermined number of randomly selected pixels. For most combinations of ONN size and coupling function, simulations were run with 0%, 12.5%, and 25% of pixels flipped. In cases where the 64 oscillator ONN produced bad results, the run with 12.5% of pixels flipped was replaced by a run with only one pixel flipped, to confirm that the network is fully incapable of performing an associative memory task in that configuration. The results for the cases with linear and symmetric exponential coupling can be found in figure 5.1. The 8-oscillator system could successfully complete the task in both configurations, while the 64-oscillator system failed when it was configured to use the symmetric exponential coupling function. This failure may have been caused by the simulation setup, which failed to produce results for the 8 by 8 symmetric exponential configuration when the coupling strength was at normal levels. The results shown for this configuration were produced with a very low coupling strength. This made the system either completely synchronize, or leave the input pattern unaltered. The results for the configurations with polynomial and exponential coupling are in figure 5.2. All configurations utilizing the 3^{rd}-order polynomial coupling function completed the task successfully, despite their asymmetric responses to different phase-states. The 8 by 8 configuration with exponential coupling failed, but the outputs it produced still resemble some patterns that could plausibly be derived from the patterns that were taught it. This shows that this configuration is still capable of storing some information. Perhaps it would be capable of performing this task if a different learning rule is used, but it could also be the case that using this coupling function inherently decreases the memory capacity at larger network sizes. The exponential coupling causes no problems in the 2 by 4 network.

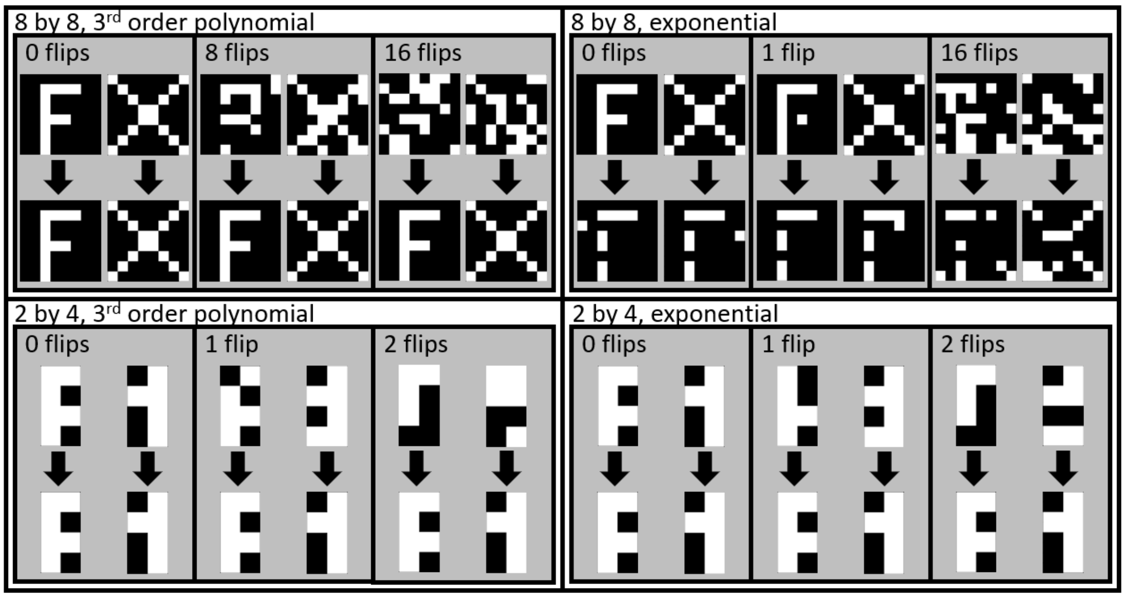

Figure 5.2: Results of the associative memory task performed with 3^{rd} order polynomial, and exponential coupling functions. The number of 'flips' indicates the hamming distance between the learnt pattern and the input.

5.2 Classification

A simulated 66 oscillator (8 by 8 image with 2 output oscillators) ONN was made to perform a classification task. The first 64 oscillators were initialized to the input image, and the 2 output

	Hamming distance:		
coupling:	0	8	16
linear	✓	✓	✓
symm. exp.	✓	✓	X
exponential	✓	✓	X
polynomial	✓	✓	✓

Table 5.1: Results of the classification task. A ✓ indicates a lack of errors after classifying at least 3 modified versions of each pattern.

oscillators were initialized to a phase of 90°. Each output oscillator corresponds to the detection of one class of image, so the possible outputs also include cases wherein both or none of the classes were detected. The coupling coefficients for classification were obtained according to the following steps:

1. Label each image that should be recognised by appending to it the desired output.
2. Teach these patterns to the network using the Storkey learning rule.
3. Remove all connections that do not directly affect the output oscillators.

The move from a single-layer fully connected network to a two-layer feedforward network occurs in the last step of this process and makes it so that this system requires at most 194 DNPUs according to equation 3.7.
The classification task was performed with the same set of different coupling functions as the associative memory task. Because the two-layer feedforward network is much less complex than the fully interconnected network used for the associative memory task, the classification task was only performed on the 8 by 8 images. The results from the task can be found in table 5.1. The system was able to perform classification using each coupling function. The error rate became noticeably higher when the systems with (symmetric) exponential coupling attempted to recognise images with a Hamming distance of 16 from the original patterns.

Chapter 6

Discussion and conclusions

The results show that the proposed architecture is capable of performing basic ONN tasks, but there are many things these results do not show well. The results do not necessarily justify using a DNPU in this architecture. In a way, the results show that an ONN can function despite the presence of some types of non-linearity (specifically 3^{rd} polynomial coupling), but not that it benefits from them at all. It does not help that these results can only serve as a proof of principle, and not an evaluation of performance.

6.1 Improving task performance measurements

Evaluating the performance of the system using a few statistically relevant metrics could shed more light on the effects of the non-linear coupling. For example, It would be very interesting to see if the system's memory capacity was affected. Measuring the system's memory capacity could be done by running numerous simulations using different sets of patterns (of different sizes too), and plotting its error rates.
In the testing done for the results, many of the input images were randomly altered versions of the patterns taught to the ONN. The ideal way to test using randomly generated inputs is to test a representative sample of them, and then report the success or failure rate. This was not done for these tests, which limits the generalizability of the results, but it nevertheless constitutes a proof of principle.

6.2 Improving the simulation setup

The possible improvements mentioned in the previous section all have one thing in common: they require a much larger sample size than has been produced for the results thus far. This requirement is the reason why they are listed as possible improvements, instead of being implemented in the results. The current simulation setup is too slow, and too unreliable, to provide the data that would be needed. Simulating the 8 by 8 fully connected associative memory task fully saturates a modern CPU for several minutes (tested on AMD Ryzen 7 2700X and Intel i7-10750H) to produce just one sample of data. This could still be fast enough if the process was properly automated, but this was not done in this case. For some combinations of configuration and coupling strength, the simulator would run as normal, but not produce any output. This problem could be avoided by turning down the coupling strength a little and running it again, but this process had not been automated.

6.3 Improving DNPU utilization

The set of tasks that were tested on the DNPU fundamentally requires only a single dimension of tuneability. This is due to these kinds of tasks being originally conceived to run in a network where the coupling elements can only change unidimensionally. These tasks still make sense to test to have a simple proof of principle but do not utilize the full computational capabilities a DNPU could provide. To justify the use of DNPUs in this architecture, more complicated tasks need to be found and tested.
One way to start exploring more complicated tasks is by eliminating the restrictions on the input and output of the system. Currently, input and output patterns are binary. This greatly limits the range of operations the ONN could perform and also limits the utility of non-linear coupling functions. Considering that the DNPU has some capability to select from the sub-harmonics in the signal it produces, it would not be inconceivable that by cleverly exploiting this one could make a network that operates with multilevel, or even continuous, phase states (See [21] for an example of a multilevel ONN).

6.4 Conclusions

In this BSc. assignment, an ONN architecture using DNPUs was described. This architecture was simulated, and a demonstration of its basic functionality as an ONN was provided. These results do not fully characterise the performance of the ONN and severely underutilize the computational capabilities of the DNPUs in the system. To further utilize the capabilities of this architecture, there is a need to look beyond the canonical ONN workloads. Some possible starting points for follow-up research have also been provided, in the form of the ODAMN.

Bibliography

[1] F. C. Hoppensteadt and E. M. Izhikevich, *Weakly Connected Neural Networks.* Springer, 1997. 2, 6, 7

[2] M. Abernot and T.-S. Aida, “Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures,” *Neuromorphic Computing and Engineering*, vol. 3, no. 1, p. 014006, Feb. 2023. [Online]. Available: https://dx.doi.org/10.1088/2634-4386/acb2ef 3, 4

[3] J. A. Acebrón, L. L. Bonilla, C. J. Pérez Vicente, F. Ritort, and R. Spigler, “The Kuramoto model: A simple paradigm for synchronization phenomena,” *Reviews of Modern Physics*, vol. 77, no. 1, pp. 137–185, Apr. 2005, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.77.137 2

[4] C. Delacour, S. Carapezzi, M. Abernot, and A. Todri-Sanial, “Energy-Performance Assessment of Oscillatory Neural Networks Based on VO 2 Devices for Future Edge AI Computing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. XX, no. X, Jan. 2023. [Online]. Available: http://www.scopus.com/inward/record.url?scp=85148421826&partnerID=8YFLogxK 2, 3, 4

[5] F. C. Hoppensteadt and E. M. Izhikevich, “Oscillatory Neurocomputers with Dynamic Connectivity,” *Physical Review Letters*, vol. 82, no. 14, pp. 2983–2986, Apr. 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.82.2983 2, 3

[6] R. W. Hölzel and K. Krischer, “Pattern recognition with simple oscillating circuits,” *New Journal of Physics*, vol. 13, no. 7, p. 073031, Jul. 2011. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/13/7/073031 2, 3, 4

[7] 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,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.713054 2

[8] P. Maffezzoni, B. Bahr, Z. Zhang, and L. Daniel, “Analysis and Design of Boolean Associative Memories Made of Resonant Oscillator Arrays,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 1964–1973, Nov. 2016. 3, 6, 7

[9] J. Shamsi, M. J. Avedillo, B. Linares-Barranco, and T. Serrano-Gotarredona, “Hardware Implementation of Differential Oscillatory Neural Networks Using VO 2-Based Oscillators and Memristor-Bridge Circuits,” *Frontiers in Neuroscience*, vol. 15, p. 674567, Jul. 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.674567/full 3

[10] K. Kostorz, R. W. Hölzel, and K. Krischer, “Distributed coupling complexity in a weakly coupled oscillatory network with associative properties,” *New Journal of Physics*, vol. 15, no. 8, p. 083010, Aug. 2013. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/15/8/083010 3, 4

[11] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982, publisher: Proceedings of the National Academy of Sciences. [Online]. Available: https://www.pnas.org/doi/10.1073/pnas.79.8.2554 3

[12] A. Storkey, "Increasing the capacity of a Hopfield network without sacrificing functionality," vol. 1327, Sep. 1999. 4

[13] D. Krotov and J. J. Hopfield, "Dense Associative Memory for Pattern Recognition," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://papers.nips.cc/paper_files/paper/2016/hash/eaae339c4d89fc102edd9dbdb6a28915-Abstract.html 4, 6, 7

[14] M. Demircigil, J. Heusel, M. Löwe, S. Upgang, and F. Vermet, "On a model of associative memory with huge storage capacity," *Journal of Statistical Physics*, vol. 168, no. 2, pp. 288–299, Jul. 2017, arXiv:1702.01929 [math]. [Online]. Available: http://arxiv.org/abs/1702.01929 4, 7

[15] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, T. Adler, L. Gruber, M. Holzleitner, M. Pavlović, G. K. Sandve, V. Greiff, D. Kreil, M. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter, "Hopfield Networks is All You Need," Apr. 2021, arXiv:2008.02217 [cs, stat]. [Online]. Available: http://arxiv.org/abs/2008.02217 4

[16] T. Chen, J. van Gelder, B. van de Ven, S. V. Amitonov, B. de Wilde, H.-C. Ruiz Euler, H. Broersma, P. A. Bobbert, F. A. Zwanenburg, and W. G. van der Wiel, "Classification with a disordered dopant-atom network in silicon," *Nature*, vol. 577, Jan. 2020. [Online]. Available: https://www.nature.com/articles/s41586-019-1901-0 5, 7

[17] H.-C. Ruiz-Euler, U. Alegre-Ibarra, B. v. d. Ven, H. Broersma, P. A. Bobbert, and W. G. v. d. Wiel, "Dopant network processing units: towards efficient neural network emulators with high-capacity nanoelectronic nodes," *Neuromorphic Computing and Engineering*, vol. 1, no. 2, p. 024002, Sep. 2021, publisher: IOP Publishing. [Online]. Available: https://dx.doi.org/10.1088/2634-4386/ac1a7f 5

[18] H.-C. Ruiz Euler, M. N. Boon, J. T. Wildeboer, B. van de Ven, T. Chen, H. Broersma, P. A. Bobbert, and W. G. van der Wiel, "A deep-learning approach to realizing functionality in nanoelectronic devices," *Nature Nanotechnology*, vol. 15, no. 12, pp. 992–998, Dec. 2020, number: 12 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41565-020-00779-y 5

[19] H. Tertilt, J. Bakker, M. Becker, B. de Wilde, I. Klanberg, B. J. Geurts, W. G. van der Wiel, A. Heuer, and P. A. Bobbert, "Hopping-Transport Mechanism for Reconfigurable Logic in Disordered Dopant Networks," *Physical Review Applied*, vol. 17, no. 6, p. 064025, Jun. 2022, publisher: American Physical Society. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.17.064025 5

[20] N. Brum, "PyLTSpice PyPI." [Online]. Available: https://pypi.org/project/PyLTSpice/ 10

[21] A. Velichko, M. Belyaev, and P. Boriskov, "A Model of an Oscillatory Neural Network with Multilevel Neurons for Pattern Recognition and Computing," *Electronics*, vol. 8, no. 1, p. 75, Jan. 2019, arXiv:1806.03079 [nlin]. [Online]. Available: http://arxiv.org/abs/1806.03079 16