



# **Nemo Project**

## **NemoSim Simulator Tool User Guide**

**Rev. 1.1**

**August 2025**



## Documentation Control

### *History Table*

Version	Date	Description
1.0	31 July 2025	Initial release
1.1	7 August 2025	Revised and expanded

## Disclaimer and Proprietary Information Notice

The information contained in this document does not represent a commitment on any part by Ceva, Inc., or its subsidiaries (collectively, "Ceva"). Ceva makes no warranty of any kind with regard to this material, including, but not limited to implied warranties of merchantability and fitness for a particular purpose whether arising out of law, custom, conduct, or otherwise.

Additionally, Ceva assumes no responsibility for any errors or omissions contained herein, and assumes no liability for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, fees, or expenses, of any nature or kind, which are incurred in connection with the furnishing, performance, or use of this material.

This document contains proprietary information, which is protected by U.S. and international copyright laws. All rights reserved. No part of this document may be reproduced, photocopied, or translated into another language without the prior written consent of Ceva. CEVA® is a registered name of Ceva. All product names are trademarks of Ceva, or of its applicable suppliers if so stated.

## Support

Ceva makes great efforts to provide a user-friendly software and hardware development environment. Along with this, Ceva provides comprehensive documentation, enabling users to learn and develop applications on their own. Due to the complexities involved in the development of DSP applications that might be beyond the scope of the documentation, an online Technical Support Service has been established. This service includes useful tips and provides fast and efficient help, assisting users to quickly resolve development problems.

### How to Get Technical Support:

- **FAQs:** Visit our website <http://www.ceva-ip.com> or your company's protected page on the Ceva website for the latest answers to frequently asked questions.
- **Application Notes:** Visit our website <http://www.ceva-ip.com> or your company's protected page on the Ceva website for the latest application notes.
- **Email:** Use the Ceva central support email address [ceva-support@ceva-ip.com](mailto:ceva-support@ceva-ip.com). Your email will be forwarded automatically to the relevant support engineers and tools developers who will provide you with the most professional support to help you resolve any problem.
- **License Keys:** Refer any license key requests or problems to [sdtkeys@ceva-ip.com](mailto:sdtkeys@ceva-ip.com). For SDT license keys installation information, see the *SDT Installation and Licensing Scheme Guide*.

**Email:** [ceva-support@ceva-ip.com](mailto:ceva-support@ceva-ip.com)

**Visit us at:** [www.ceva-ip.com](http://www.ceva-ip.com)

## Table of Contents

<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1 Overview .....	1
1.2 Purpose .....	1
1.3 Key Features.....	1
1.4 Typical Use Cases .....	2
<b>2. USING THE NEMOSIM SIMULATION TOOL.....</b>	<b>3</b>
2.1 Step 1: Preparing the XML/TXT Files .....	4
2.2 Step 2: Preparing the JSON File.....	8
2.3 Step 3: Running the NemoSim Simulator Tool .....	9
2.4 Step 4: Plotting the Outputs.....	10
<b>3. ERROR HANDLING.....</b>	<b>11</b>
3.1 Error and Warning Message Formats .....	11
3.2 Possible Return Codes .....	11
<b>4. REFERENCES .....</b>	<b>13</b>

## List of Examples

Example 2-1: NemoSim Progress Messages.....	9
---	---

## List of Tables

Table 2-1: LIF Network XML Configuration Parameter Definitions .....	5
Table 2-2: BIU Network XML Configuration Parameter Definitions.....	6

# 1. Introduction

## 1.1 Overview

This document describes the NemoSim simulation tool, which is used for spiking neural network architectures, and has been developed as part of the Nemo Project.

It enables researchers and engineers to model, simulate, and analyze the behavior of various neural network types, such as Leaky Integrate-and-Fire (LIF) and Brain-Inspired Unit (BIU) networks.

## 1.2 Purpose

NemoSim is designed to:

- Simulate neural network architectures using user-defined configurations
- Model time-evolving neuron and synapse states in response to input currents
- Support both standard and custom network types (LIF, BIU, and extensible for new models)
- Generate outputs for scientific analysis and visualization, including detailed time-series data for neural state, synapse activity, and spikes
- Facilitate debugging and research, with outputs and tools to identify and analyze network behavior

## 1.3 Key Features

- **Flexible Input:** NemoSim accepts XML-based network descriptions and plain text current input files, supporting both manual and scripted generation.
- **Extensible Design:** The modular codebase allows for easy addition of new neuron/synapse models and architectures.
- **Comprehensive Output:** The tool produces multiple output files (membrane potentials, synaptic inputs, spikes, and so on) for analysis and plotting.
- **Analysis Tools:** NemoSim includes Python scripts and guidelines for visualizing and interpreting simulation results.
- **Robust Error Handling:** The tool detects and reports errors in input files and configuration, aiding reproducibility.

## 1.4 Typical Use Cases

- Computational neuroscience research
- Neuromorphic hardware prototyping and validation
- Algorithm development and prototyping for spiking neural networks



## 2. Using the NemoSim Simulation Tool

To use the NemoSim simulation tool, do the following:

1. Prepare the XML/TXT files, as described in Section 2.1
2. Prepare the JSON file, as described in Section 2.2
3. Run the NemoSim tool, as described in Section 2.3
4. Plot the output, as described in Section 2.4

## 2.1 Step 1: Preparing the XML/TXT Files

Do the following:

1. In an XML configuration file, define your network architecture, neuron parameters, and simulation settings.

- Important:**
- *The XML file must have a `<NetworkConfig>` root with a `type` attribute.*
  - *Required child elements depend on the network type (LIF, BIU, and so on).*
  - *All required numeric fields must be present and valid numbers.*

For example:

- For a LIF network:

```
<NetworkConfig type="LIF">
  <LIFNetwork>
    <Cm>1.0</Cm>
    <Cf>0.5</Cf>
    <VDD>2.5</VDD>
    <!-- other LIF parameters -->
  </LIFNetwork>
  <Architecture>
    <!-- network connectivity, layers, weights, etc. -->
  </Architecture>
</NetworkConfig>
```

Table 2-1 defines the key parameters that are used in this section of the LIF XML configuration file.

These values are based on standard analog neuron circuit modeling practices and can be adapted depending on the simulation context.

**Table 2-1: LIF Network XML Configuration Parameter Definitions**

Parameter	Description	Unit	Example Value	Notes
Cm	Membrane capacitance	Farads (F)	1e-6	Determines the neuron integration time constant
Cf	Feedback capacitance	Farads (F)	1e-9	Affects how quickly the membrane voltage resets or leaks after a spike
VDD	Supply voltage	Volts (V)	5.0	Sets the upper voltage bounds for circuit behavior
VTh	Threshold voltage	Volts (V)	1.0	Defines the spike generation threshold
K	Gain factor	N/A	2	Dimensionless scaling factor (for example, for current mirror)
Rmin/Rmax	Min/Max resistance in synaptic array	Ohms ( $\Omega$ )	145e3/145e6	Used for dynamic range in weight representation
gm	Transconductance	Siemens (S)	6.896e-6	Typically from the input differential pair
CGB*, CGD*, CDB*	Parasitic capacitances (Gate-*, Drain-*, Bulk-*)	Farads (F)	~e-18 values	Derived from transistor model extraction
req	Effective resistance	Ohms ( $\Omega$ )	145e3	Used in equivalent RC modeling
R_da	Driver array resistance	Ohms ( $\Omega$ )	10e3	Series resistance from driver circuitry

- For a BIU network:

```
<NetworkConfig type="BIU">
  <BIUNetwork>
    <!-- BIU parameters here -->
  </BIUNetwork>
  <Architecture>
    <!-- architecture details -->
  </Architecture>
</NetworkConfig>
```

Table 2-2 defines the key parameters that are used in this section of the BIU XML configuration file.

These values are based on a switched-capacitor SNN architecture using programmable digital weights, charge sharing, and comparator-based thresholding.

**Table 2-2: BIU Network XML Configuration Parameter Definitions**

Parameter	Description	Unit	Example Value	Notes
Cu	Synapse unit capacitance	Femtofarads (fF)	4	Defines the granularity of weight resolution
Cn	Neuron integration capacitor	Femtofarads (fF)	1000	Large capacitor for temporal integration
VDD	Supply voltage	Volts (V)	1.2	Nominal voltage for digital and analog circuitry
VTh	Comparator threshold voltage	Volts (V)	±0.4	Spike generation threshold (programmable)
WS	Weight sign	N/A	+1 or -1	<ul style="list-style-type: none"> <li>• Positive = VDD</li> <li>• Negative = VSS</li> </ul>
W[3:0]	Synaptic weight (4-bit)	Digital	0–15	Multiplies the charge contribution per synapse
NRcycles	Refractory period (cycles)	Integer	1–8	Input is blocked for N cycles after spike
Cl	Leakage capacitance (optional)	Femtofarads (fF)	5e-15	Models subthreshold leakage or passive decay
Vm	Neuron potential	Volts (V)	Dynamic	Computed via charge-sharing at each phase
Nu	Number of synaptic inputs	N/A	(varies by layer)	Determines the total Cu contribution per neuron

2. In a plain TXT file, define the neuron values.

**Important:** Each line shall contain values corresponding to the neurons in the first layer (without delimiters), representing the input current for each time step (or input channel).

For example:

```
1 0 1 0 1 1 1 0
0 0 1 0 1 1 1 0
0 0 1 0 1 1 1 0
0 0 1 0 1 1 1 0
1 0 1 0 1 1 1 0
```

**Tip:** You can use the provided Python **input\_creator.py** script to generate input files; for example:

```
python input_creator.py
> "1e-10 * math.sin(2 * math.pi * t * 5 + 3 *
math.pi/2) + 1e-10"
```

## 2.2 Step 2: Preparing the JSON File

Do the following:

- In a JSON file, define your workspace parameters and input files paths.

For example:

- For a LIF network:

```
{
  "output_directory":
    "./Tests/SNN/LIF/sin_current_test/",
  "xml_config_path":
    "./Tests/SNN/LIF/sin_current_test/testFull.xml",
  "data_input_file":
    "./Tests/SNN/LIF/sin_current_test/input.txt",
  "progress_interval_seconds": 2
}
```

- For a BIU network:

```
{
  "output_directory": "./Tests/SNN/BIU/",
  "xml_config_path": "./Tests/SNN/BIU/test.xml",
  "sup_xml_config_path":
    "./Tests/SNN/BIU/supervisor.xml",
  "data_input_file": "./Tests/SNN/BIU/input.txt",
  "progress_interval_seconds": 2
}
```

## 2.3 Step 3: Running the NemoSim Simulator Tool

Do the following:

- From the root directory, type:

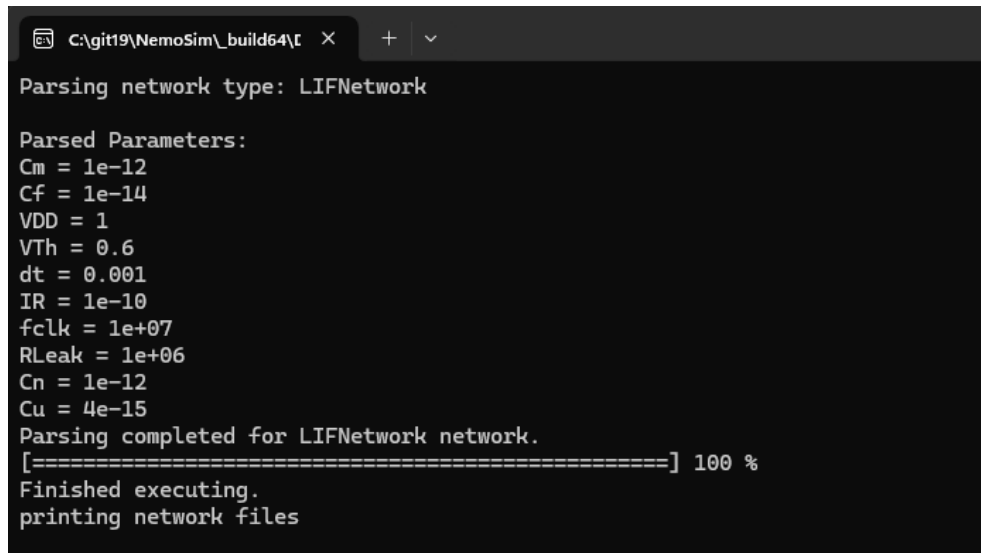
```
NEMOSIM.exe <name of JSON file>
```

For example:

```
NEMOSIM.exe config.json
```

While the NemoSim is running, it will display progress messages on the screen, as demonstrated in Example 2-1. If an error occurs, an error or warning message will be displayed (for details, see Section 3).

*Example 2-1: NemoSim Progress Messages*



```
C:\git19\NemoSim_build64\  X  +  v
Parsing network type: LIFNetwork

Parsed Parameters:
Cm = 1e-12
Cf = 1e-14
VDD = 1
VTh = 0.6
dt = 0.001
IR = 1e-10
fclk = 1e+07
RLeak = 1e+06
Cn = 1e-12
Cu = 4e-15
Parsing completed for LIFNetwork network.
[=====] 100 %
Finished executing.
printing network files
```

## 2.4 Step 4: Plotting the Outputs

When the NemoSim has finished running, it generates output files and places them in the output directory you specified in the JSON file (as described in Section 2.2).

Output files are plain text, each containing a list of numeric values (one per line). Each output file corresponds to a specific neural variable, where **<x>** is the number of the layer, and **<y>** is the number of the neuron:

- LIF Simulation (three files per neuron):
  - **Iins<x><y>.txt**: Input currents
  - **vms<x><y>.txt**: Membrane potentials
  - **Vouts<x><y>.txt**: Output voltages (spikes)
- BIU Simulation (three files per neuron):
  - **Vin<x><y>.txt**: Synapse input values
  - **Vns<x><y>.txt**: Neural state potentials
  - **Spikes<x><y>.txt**: Output spikes

After the output files have been generated, you can plot or analyze them to look for neurons with unexpected behaviors (for example, constant potentials or no spikes) for further debugging.

Do one of the following:

- For LIF networks, use the **plot\_vm\_to\_dt.py** script.
- For BIU networks, use the **plot\_vn\_to\_dt.py** script.

For example:

```
python plot_vm_to_dt.py Iins00.txt vms00.txt Vouts00.txt
```



## 3. Error Handling

### 3.1 Error and Warning Message Formats

- Errors (examples):

```
Error loading XML file: <description>
Error: No <NetworkConfig> root element found.
Error: Network type attribute not found in
<NetworkConfig>.
```

- Warnings:

```
Warning: <Cm> missing or invalid in LIFNetwork
```

### 3.2 Possible Return Codes

The code uses TinyXML2, which defines error codes such as:

- XML\_SUCCESS (0): Success
- XML\_NO\_ATTRIBUTE
- XML\_WRONG\_ATTRIBUTE\_TYPE
- XML\_ERROR\_FILE\_NOT\_FOUND
- XML\_ERROR\_FILE\_COULD\_NOT\_BE\_OPENED
- XML\_ERROR\_FILE\_READ\_ERROR
- XML\_ERROR\_PARSING\_ELEMENT

For the full list, see TinyXML2's **XML\_Error** enum.



## 4. References

- For details and example files for LIF, see **Tests/SNN/LIF/**.
- For details and example files for BIU, see **Tests/SNN/BIU/**.
- Output file examples: **Vin.txt**, **Vns.txt** (in the test folders).