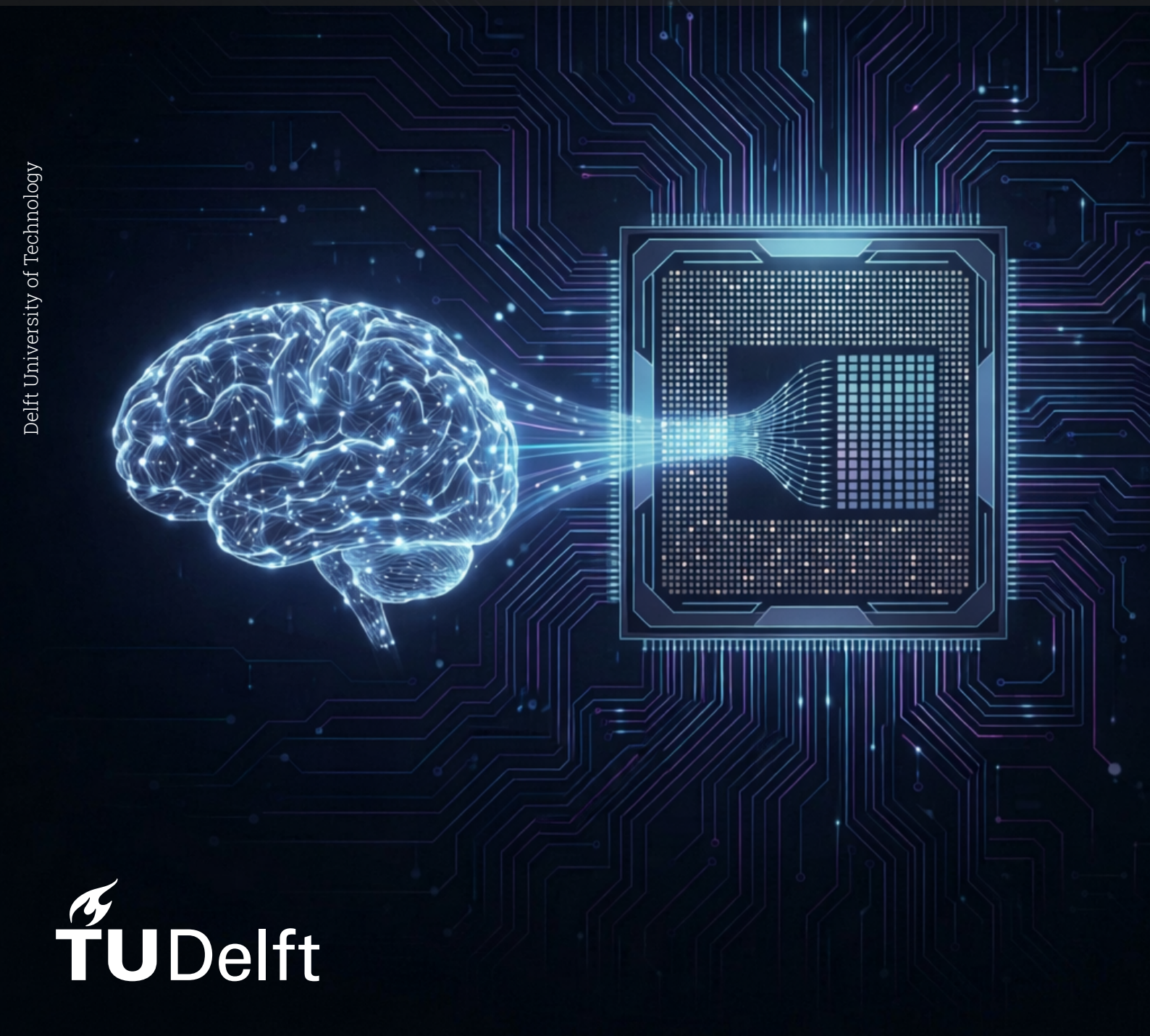
Real-Time Brain Simulation on Resource-Constrained FPGAs Using Neural Network-Based LUT Compression

Low-resource Hodgkin-Huxley Model Simulation

Maxim Mazurovs



Delft University of Technology

TU Delft

Real-Time Brain Simulation on Resource-Constrained FPGAs Using Neural Network-Based LUT Compression

Low-resource Hodgkin-Huxley Model Simulation

to obtain the degree of Master's of Science
at Delft University of Technology,
to be defended in public on Monday, July 7th, 2026, at 13:00 a.m.

by

Maxim Mazurovs

Student Computer and Embedded-Systems Engineering, TU Delft,
in collaboration with the Neuroscience Department, Erasmus MC

Student number:	5050545	
Project duration:	September 1, 2025 - July 7, 2026	
Thesis Committee:	Dr.ir. J.S.S.M. Wong	TU Delft, chair
	Prof.Dr.ir. W.A. Serdijn	TU Delft
	Dr.ir. C. Strydis	TU Delft, supervisor
	MSc. L.P.L. Landsmeer	TU Delft, co-supervisor

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
Thesis cover generated by Gemini AI

TU Delft Delft University of Technology

NCL NEURO COMPUTING LAB

“If the human brain were so simple that we could understand it,
we would be so simple that we couldn’t.”

Emerson M. Pugh

Summary

Real-time neuromorphic simulation of biophysically accurate neuron models, such as the Hodgkin-Huxley (HH) model, is crucial for advanced neuroprosthetic applications, such as digital brain twins. However, existing FPGA implementations face a severe trade-off: they either achieve high biological fidelity with resource-intensive designs that offer low neuron density per unit area, or achieve large-scale network scalability by relying on simplified models that sacrifice biological fidelity. This limitation stems from the computational overhead of evaluating the nonlinear ion-channel equations of the HH model, which traditionally requires costly arithmetic operations and dedicated hardware resources. To address this, there is a critical motivation to develop high-density architectures that maximise neuron count within a fixed FPGA resource budget by transforming non-linear computational components toward optimised logic structures.

This thesis exploits recent advances in neural network-based lookup table (NeuraLUT) generation, specifically utilising Brevitas' Quantisation-Aware Training (QAT) approach to map decomposed complex HH functions into compact hardware structures. Rather than relying on traditional analytical approximations that sacrifice accuracy and fidelity, this methodology trains deep learning models to represent the non-linear ion-channel dynamics and compresses them into highly simplified lookup tables (LUTs) using adapted versions of NeuraLUT and ReducedLUT tools. This conversion enables the FPGA to evaluate the learned non-linear functions directly through its logic-based lookup tables at runtime, eliminating the need for explicit neural network inference or costly arithmetic evaluation.

To realise this framework, four distinct hardware architectures, varying in the granularity of the HH model's structural decomposition, were explored alongside multiple optimisations, including LUT compression, derivative-based predictions, neural-network-based LUT generation, and bit-resolution reduction. The final selected design configuration, referred to as BrainLUT, achieves the most favourable trade-off between stable neuron dynamics and resource utilisation.

The resulting proof-of-concept neuronal network successfully simulates 1250 neurons in real-time on an Artix-7 XC7A200T device at an operating frequency of 125 MHz, consuming only 6307 LUTs (4.71% utilisation), 15 BRAM blocks (3.01% utilisation), and 4 DSP blocks (0.54% utilisation). Although this does not represent the highest reported neuron count in the literature, it demonstrates superior resource efficiency and per-neuron density compared with existing FPGA-based HH implementations, establishing a highly resource-efficient foundation for future scaling. The achieved resource headroom and modular hardware architecture indicate that the proposed design can be further scaled by replicating the neuron pipelines, enabling substantially larger real-time neuronal networks while maintaining biophysical model fidelity for future neuromorphic applications.

Preface

I would like to thank my supervisors, Christos Strydis and Lennart Landsmeer, for giving me the opportunity to work on this topic and for their guidance throughout this project. I am especially grateful for the discussions, feedback, and support that helped me navigate the challenges of this work and develop a broader perspective on the research. I also appreciate the trust and encouragement to continuously improve the quality of my work and pursue the best possible version of this research.

I would also like to thank Mario Negrello for the insightful conversations that extended beyond the direct scope of this project and broadened my understanding of neural and neuronal networks. Furthermore, I would like to thank Georgios Smaragdos for sharing his expertise, providing valuable feedback on my thesis, and helping me explore related work. Additionally, I would like to thank the students and researchers at NCL (Erasmus MC) for the supportive environment, the weekly discussions, and the many interesting conversations throughout this journey. The interactions and friendships formed along the way made this experience both enjoyable and memorable.

Outside of the academic environment, I would like to thank my roommates at Huize Avondlust for the countless shared meals and the support they provided both day and night. I am also grateful to my friends at AEGEE-Delft for the much-needed moments of distraction and relaxation. In particular, I would like to thank Eva Groenewoud and Daniël Blom for the many long study sessions, motivating conversations, and the hours spent working together at Echo and Fellowship at the end of my thesis. I would also like to thank my fellow board members from the 150th Board of the ETV for the unforgettable experiences, the moments that helped me step away from thesis life, and the support and motivation we shared along the way. Furthermore, I would like to thank everyone from the different ETV Boards throughout my student years, from the 146th to the 154th Board, as well as all other ETV members who have been part of my student journey, especially those graduating alongside me. In particular, I would like to thank Simon Molenkamp for the many meaningful conversations we shared, ranging from relieving frustrations to discussing the technical challenges of our theses, as well as for his valuable feedback during the final stages of my thesis.

Finally, I would like to thank everyone at ETV and AEGEE-Delft who has been part of my life over the past seven years. The friendships, conversations, activities, trips, and countless memorable moments have made my student years truly special, and I cannot imagine this period without the people I met through these associations.

Last but not least, I would like to thank my family and friends. I am deeply grateful to my parents, who have always supported me throughout my life and have been an example of perseverance and dedication in pursuing my goals. I am incredibly thankful for everything they have done, and continue to do, for me. I would also like to thank my two little sisters, who bring so much joy into my life and with whom I look forward to spending more time again after finishing this thesis. Furthermore, I would like to thank my extended family and stepfamilies, who, despite being spread across different countries and often far away, have always remained close to my heart and have always been there for me. Their support, care, and presence have meant a great deal to me throughout my life. Finally, I would like to thank my dear friends from high school, especially Jochem van de Kamp and Emil de Jong, who have always been there for me throughout the many ups and downs of life.

I dedicate this thesis to those affected by brain injuries who have not been as fortunate in their recovery as I have been. I hope this work may serve as a small step in neuromorphic computing, contributing to future developments in neuromorphic systems and neuroprosthetic technologies that may one day support the restoration or augmentation of functions affected by neurological injury.

Maxim Mazurovs
Delft, June 2026

Contents

Summary ii

Preface iii

List of Figures x

List of Tables xii

Nomenclature xiii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Research Question and Thesis Goals . . . 3
- 1.3 Thesis Methodology . . . 3
- 1.4 Thesis Organisation . . . 3

2 Background & Related Work 4
- 2.1 The Neuron . . . 4
  - 2.1.1 Computational Neuron Models . . . 6
  - 2.1.2 Numerical simulation of HH dynamics . . . 9
  - 2.1.3 Neuronal Network . . . 11
- 2.2 Brain Simulation on Hardware Platforms . . . 12
  - 2.2.1 FPGA Architecture . . . 12
- 2.3 Neural Network-based LUT Compression . . . 14
  - 2.3.1 Neural Networks . . . 14
  - 2.3.2 NeuraLUT . . . 16
  - 2.3.3 LUT Compression . . . 17
- 2.4 Related Work . . . 19
- 2.5 Conclusion . . . 20

3 System Design 21
- 3.1 Problem Analysis . . . 21
  - 3.1.1 Resolution . . . 21
  - 3.1.2 Functional Requirements . . . 22
- 3.2 Design Candidates . . . 24
  - 3.2.1 Design 1 [MONOLITH]: 1 unified LUT . . . 25
  - 3.2.2 Design 2 [SPLIT]: Separate Gate LUTs and Voltage LUT . . . 27
  - 3.2.3 Design 3 [IONIC-SUM]: gate LUTs and summed-current LUTs . . . 29
  - 3.2.4 Design 4 [HYBRID]: separate gate and conductance LUTs + some multiplications 31
- 3.3 Design Evaluation . . . 33

4 System Implementation 34
- 4.1 Overview . . . 34
- 4.2 First Optimisation: LUT compression . . . 35
  - 4.2.1 Optimisation 1: Design 4 [HYBRID] . . . 35
  - 4.2.2 Optimisation 1: Design 3 [IONIC-SUM] . . . 36
  - 4.2.3 Optimisation 1: Requirement Check . . . 37
- 4.3 Second Optimisation: Delta Prediction . . . 37
  - 4.3.1 Optimisation 2: Design 4 [HYBRID] . . . 38
  - 4.3.2 Optimisation 2: Design 3 [IONIC-SUM] . . . 39
  - 4.3.3 Optimisation 2: Requirement Check . . . 40
- 4.4 Third Optimisation: NeuraLUT . . . 41

4.4.1 Optimisation 3: Design 1 [MONOLITH] . . . 41
4.4.2 Optimisation 3: Design 4 [HYBRID] . . . 42
4.4.3 Control Experiment: Without Delta Prediction . . . 43
4.4.4 Optimisation 3: Requirement Check . . . 44
4.5 Fourth Optimisation: Bit-Resolution Reduction . . . 44
4.5.1 Optimisation 4: Design 4 [HYBRID] . . . 45
4.5.2 Optimisation 4: Design 3 [IONIC-SUM] . . . 45
4.5.3 Optimisation 4: Design 2 [SPLIT] . . . 46
4.5.4 Optimisation 4: Requirement Check . . . 47
4.6 Optimisation Evaluation . . . 47

5 Simulation Framework and Algorithm Optimisation 49
5.1 Python: HH Simulation . . . 50
5.1.1 Dataset Generation . . . 50
5.2 Brevitas: Neural Network Training . . . 51
5.2.1 Normalisation . . . 51
5.2.2 Quantisation-Aware Training (QAT) . . . 51
5.2.3 Network Architecture and Forward Propagation . . . 52
5.2.4 Training Methodology and Parameter Tuning . . . 54
5.2.5 Evaluation: Autoregressive Testing . . . 56
5.3 NeuraLUT: Conversion to Hardware . . . 57
5.3.1 Adaptation 1: Chunked Truth-Table Generation . . . 57
5.3.2 Adaptation 2: Direct LUT Addressing . . . 57
5.3.3 Adaptation 3: Parallelised Verilog Generation . . . 58
5.4 ReducedLUT: LUT Compression . . . 58
5.4.1 Adaptation 1: Compact Storage of 'care' values . . . 59
5.4.2 Adaptation 2: Subtable Pattern Matching . . . 59
5.4.3 Adaptation 3: Noise Safety Margin . . . 60
5.4.4 Adaptation 4: Pre-shifted Final Data for Synthesis . . . 61
5.4.5 ReducedLUT Parallelisation . . . 62
5.4.6 Discussion on Adaptation Performance . . . 62
5.5 Verilog: Neuron Core . . . 62
5.5.1 Pipeline . . . 62

6 Neuronal Network 64
6.1 Top-Level Architecture . . . 64
6.2 Neuron-Core Wrapper . . . 65
6.2.1 State-Accumulator Resolution . . . 67
6.2.2 Stimulus Exponential Decay . . . 67
6.3 Synaptic Connectivity . . . 68
6.4 Spike Router . . . 69
6.5 Spike Overflow . . . 71

7 Evaluation 74
7.1 Experimental Setup . . . 74
7.2 Neuron-Core Results . . . 75
7.2.1 Optimisation 1: LUT Compression . . . 75
7.2.2 Optimisation 2: Delta Prediction . . . 77
7.2.3 Optimisation 3: NeuraLUT . . . 80
7.2.4 Optimisation 4: Bit-Resolution Reduction . . . 86
7.2.5 Overview Neuron-Core Results . . . 92
7.3 Neuronal Network . . . 93
7.3.1 Network with Design 4 [HYBRID] . . . 93
7.3.2 Network with Design 3 [IONIC-SUM] . . . 95
7.3.3 Timing Performance . . . 96
7.3.4 Network Evaluation . . . 97
7.4 Performance Comparison . . . 98

8 Conclusions **101**
8.1 Main Contributions . . . 102
8.2 Future Work . . . 102
8.2.1 Neuron-Core Suggestions . . . 102
8.2.2 Neuronal-Network Future Work Directions . . . 103

References **104**

A ReducedLUT - Adaptation **109**

B Simulation results **111**
B.1 Dataset . . . 111
B.1.1 Training Dataset . . . 111
B.1.2 Validation Dataset . . . 112
B.2 Gate Channel Normalisation Accuracy . . . 112
B.3 Reduced Bit Resolution . . . 113
B.4 Neural Network Training Simulation . . . 114
B.4.1 Optimisation 3: Gate NN . . . 114
B.4.2 Optimisation 4: Gate NN . . . 116
B.4.3 Optimisation 4: IK and INa NN . . . 117
B.4.4 Optimisation 4: Voltage NN . . . 118
B.5 Neuronal Network . . . 120
B.5.1 Design 3 . . . 120

List of Figures

2.1 Propagation of action potentials in axons, adapted from [20] . . . 4
2.2 Illustration of ion concentration inside and outside the neuron membrane in the resting state, adapted from the illustration created by Jim Hutchins and Alexa Crookston with BioRender [21] . . . 5
2.3 Illustration of ion flow across the neuron membrane for the four different phases of an action potential . . . 5
2.4 Illustration of a neuronal action potential with annotations for each changing phase upon arrival of a stimulus signal . . . 6
2.5 Overview of different neuron models with respect to computational complexity (x-axis) and level of biological plausibility (y-axis), interpreted here in terms of spiking behaviour, adapted from [22] . . . 7
2.6 Illustration of ion channels for sodium and potassium flow with their associated gates n,m and h, and the leak channel, adapted from [25] . . . 7
2.7 Illustration of the temporal change of the ion gate channels n,m and h with respect to the action potential (V_m) . . . 8
2.8 Electrical circuit representation of the Hodgkin-Huxley model, adapted from [26] . . . 8
2.9 Illustration of the numerical solution of a differential equation using the forward Euler method, adapted from Figure 10.1.1 in [27] . . . 10
2.10 Schematic representation of the balanced excitatory-inhibitory network introduced by Brunel [28]. . . . 11
2.11 Comparison between four platforms, CPU-GPU-FPGA-ASIC, for Spiking Neural Network (SNN) applications, adapted from [33] . . . 12
2.12 Basic FPGA structure, reproduced from Fig.1.1 in [35]. . . . 13
2.13 Overview of a Configurable Logic Block, adapted from Figure 1a in [37]. . . . 13
2.14 Internal structure and operation of a lookup table (LUT), adapted from the RapidWright documentation [38]. . . . 14
2.15 Detailed illustration of a perceptron in an artificial neural network, reproduced from [43]. 15
2.16 Detailed illustration of a feed-forward neural network with 1 hidden layer, reproduced from [44]. . . . 16
2.17 Visualisation of NeuraLUT's toolflow, adapted from Figure 4 in [17]. . . . 16
2.18 Visualisation of ReducedLUT's toolflow adapted from Figure 2 in [18]. . . . 18
2.19 Visualisation of ReducedLUT's functionality of subtable matching and marking dontcare values as wildcards, adapted from Figure 1 in [18]. . . . 18

3.1 Simplified quantisation inaccuracy example showcasing how an insufficient resolution can lead to a staircase-like approximation . . . 22
3.2 Overview of the four neuron-core architectures, illustrating the decomposition progression, from the coarse-grained 'MONOLITH' architecture to the progressively fine-grained 'SPLIT', 'IONIC-SUM', and 'HYBRID' architectures . . . 25
3.3 Design 1 [MONOLITH]: 1 unified LUT mapping all possible inputs of the HH model V, n, m, h, I to the outputs that are needed in the next step's update V, n, m, h . . . 25
3.4 Visual example of the LUT designed for Design 1 (MONOLITH) . . . 26
3.5 Design-space position of Design 1 (MONOLITH) across neuron core metrics . . . 26
3.6 Design 2 [SPLIT]: separation of gate-channel LUTs and voltage-membrane LUT . . . 27
3.7 Design-space position of Design 2 (IONIC-SUM) across neuron core metrics . . . 28
3.8 Design 3 [IONIC-SUM]: separate LUTs for gate-channel dynamics and ionic-current contributions, which are summed to determine the membrane-voltage update . . . 30
3.9 Design-space position of Design 3 (IONIC-SUM) across neuron core metrics . . . 30

3.10 Design 4 [HYBRID]: two-dimensional LUTs for gate channel dynamics, one-dimensional conductance LUTs for $\bar{g}_K n^4$ and $\bar{g}_{Na} m^3$ combined with some arithmetic operations, and a leakage current I_{leak} LUT with already embedded timescaling . . . 32
3.11 Design-space position of Design 4 (HYBRID) across neuron core metrics . . . 32
3.12 Simple overview of the four neuron-core architectures supplementary to Table 3.2 . . . 33

4.1 Evaluation of the baseline across neuron core metrics . . . 35
4.2 Limit cycles V-n, V-m, V-h across 500 ms of various injected stimuli . . . 36
4.3 Next-state mapping of the gate 'n' channel before and after delta-prediction optimisation (2) . . . 38
4.4 Simplified block overview of Design 4 [HYBRID]: reformulated LUTs for the 3 gate blocks implementing delta prediction and Euler propagation, whereafter the voltage is calculated with simple LUTs and arithmetic . . . 38
4.5 Design 3 [IONIC-SUM] with derivative-predicting gate-channel LUTs and Euler-based next state update . . . 39
4.6 Simulated voltage derivatives (dV/dt) over time to visualise the small active regions and the maximum derivative value of around 450mV . . . 40
4.7 Design 1 [MONOLITH]: Neural Network-based LUTs with Euler propagation . . . 42
4.8 Design 4 [HYBRID]: compressed, Neural Network-based gate-channel LUTs with Euler propagation, low-dimensional conductance LUTs and multiplications (DSP) . . . 42
4.9 Visual example of the control experiment for optimisation 3 (NeuraLUT) without optimisation 2 (Delta Prediction) . . . 43
4.10 Design 4 [HYBRID]: Neural-Network-generated gate-channel LUTs without Delta Prediction . . . 43
4.11 Design 1 [MONOLITH]: Neural-Network-generated monolithic state-transition mapping without Delta Prediction . . . 44
4.12 Design 3 [IONIC-SUM]: reduced-precision Neural Network-generated gate-channel and conductance-current LUTs, with the leak-current contribution as a direct 1-dimensional LUT . . . 46
4.13 Design 2 [SPLIT]: reduced-precision Neural Network-generated gate-channel LUTs together with a Neural Network-generated voltage-membrane LUT replacing the original high-dimensional mapping . . . 47
4.14 Evaluation of the design architectures (a) and of the optimisations (b) across the Neuron Core metrics. Optimisations are evaluated primarily based on Design 4 [HYBRID]. . . . 48

5.1 Visualisation of the full workflow designed in this thesis, starting from HH equations to the synthesis of the neuron core and network on an FPGA . . . 49
5.2 Stimulus currents used for HH simulation . . . 50
5.3 Visualisation of the transformation from a dense fully-connected network to a sparse, pruned-connectivity network, replicated from Figure 5 in [61] . . . 53
5.4 Visual representation of the hidden sub-MLP structure within a perceptron, rebranding it to an 'L-LUT', adapted from Figure 4 in [17] . . . 53
5.5 Sub-process visualisation of the adapted ReducedLUT algorithm showcasing the subtable pattern matching (stage 2). Full process can be found in Appendix Chapter A. . . . 60
5.6 Visualisation of the value-space of the gate channel variables, showing the limited region of the 2^{24} input space that is actually visited, in combination with a margin of 50 quantisation steps around it, which was used as the *care* mask in the ReducedLUT algorithm. . . . 60
5.7 Visualisation of the 6 pipeline stages of the neuron core with Design 4 [HYBRID] implemented in hardware (Verilog), where the blocks for the gate-channels represent a threefold amount of blocks for each of the gate channels n,m, and h . . . 63

6.1 Schematic high-level representation of the top-level architecture in hardware, featuring the four main components: neuron-core wrapper, spike router, 6 different RAMs for neuron-state and synapse storage, and a unified Stim RAM Controller . . . 65
6.2 Functionality and dataflow overview of the *Stim RAM Controller*, including two alternating stimulus RAMs (active/inactive) and the stimulus decay RAM . . . 66

6.3 Schematic of the split compact representation of the synaptic connections between neurons using 2 BRAMs . . . 68
6.4 Simplified visualisation of the event-driven synaptic update mechanism after a neuron has spiked . . . 69
6.5 Architectural block diagram of the Spike Router, as part of the neuronal network architecture . . . 70
6.6 Architectural block diagram of the Spike FSM, as part of the Spike Router architecture . 70
6.7 Architectural block diagram of the Spike Pipeline, as part of the Spike Router architecture 71
6.8 Simplistic example of a synaptic update overflowing the current simulation timestep . . 72
6.9 Monte Carlo estimation of spike router overflow probability for varying network sizes and connectivity densities . . . 73

7.1 Simulation results of Design 4 [HYBRID] with LUT compression evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). . . . 76
7.2 Simulation results of Design 3 [IONIC-SUM] with LUT compression evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). . . . 77
7.3 Simulated representation of the relation between the gate channel derivatives and the membrane voltage (left) and the gate-channel behaviour across 500 ms with various stimuli applied (right) . . . 78
7.4 Simulation results of Design 4 [HYBRID] with LUT compression and delta prediction evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). . . . 79
7.5 Simulation results of Design 3 [IONIC-SUM] with LUT compression and delta prediction evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). . . . 79
7.6 Autoregressive simulation results of Design 4 [HYBRID]with NeuraLUT-based gate-channel LUTs evaluated in software . . . 80
7.7 Autoregressive simulation results of Design 1 [MONOLITH]with NeuraLUT-based LUTs evaluated in software . . . 82
7.8 Autoregressive simulation results of Design 1 [MONOLITH]with NeuraLUT-based gate-channel LUTs evaluated in software, but without the delta prediction approach as a control experiment . . . 83
7.9 One-step-prediction simulation results of Design 1 [MONOLITH]with NeuraLUT-based gate-channel LUTs evaluated in software, but without the delta prediction approach as a control experiment . . . 83
7.10 Control Experiments of Optimisation 3 (NeuraLUT) without optimisation 2 (delta prediction), part 1/2 . . . 84
7.11 Control Experiments of Optimisation 3 (NeuraLUT) without optimisation 2 (delta prediction), part 2/2 . . . 85
7.12 Comparison between three simulations that illustrate the effect of a reduced bit resolution on the LUT on which only Optimisations 1 and 2 have been applied; from pane (a) to (c) comparing between 12 bits, 9 bits and 6 bits . . . 86
7.13 Autoregressive simulation results of Design 4 [HYBRID]with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software . . . 87
7.14 Autoregressive simulation results of Design 3 [IONIC-SUM]with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software . . . 89

7.15 Autoregressive simulation results of Design 3 [IONIC-SUM]with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software on the full training dataset for detailed observation . . . 90
7.16 Autoregressive simulation results of Design 2 [SPLIT]with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software on the full training dataset for detailed observation . . . 91
7.17 Neuronal network simulation using the Design 4 [HYBRID]core for 1,250 neurons at real-time . . . 94
7.18 Neuronal network simulation using the Design 3 [IONIC-SUM]core for 1,250 neurons at real-time . . . 96
7.19 Visual comparison of the proposed work and related works based on the normalised Logic and DSP count per neuron . . . 99

A.1 Part 1 of the adaptation of the ReducedLUT algorithm. . . . 109
A.2 Part 2 of the adaptation of the ReducedLUT algorithm. . . . 110
A.3 Part 3 of the adaptation of the ReducedLUT algorithm. . . . 110

B.1 Simulated Hodgkin-Huxley model showcasing the evolution of all intermediate and resulting parameters as a result of the stimulus input from Figure 5.2a . . . 111
B.2 Simulated Hodgkin-Huxley model showcasing the evolution of all intermediate and resulting parameters as a result of the stimulus input from Figure 5.2b . . . 112
B.3 Simulated LUT output of gate m channel LUT with a resulting plot from the quantised derivative output, dm/dt normalised by -1..+1 . . . 112
B.4 Simulated LUT output of gate m channel LUT with a resulting plot from the quantised derivative output, dm/dt normalised by -3..+3 . . . 112
B.5 12-bit resolution . . . 113
B.6 10-bit resolution . . . 113
B.7 6-bit resolution . . . 114
B.8 Simulated NN autoregressive simulation gate channel n, 12 bits, normalised in own range 114
B.9 Simulated NN autoregressive simulation gate channel m, 12 bits, normalised in own range 115
B.10 Simulated NN autoregressive simulation gate channel h, 12 bits, normalised in own range 115
B.11 Simulated NN autoregressive simulaiton gate channel n, 6-6-12 bits, normalised 0..1 . . 116
B.12 Simulated NN autoregressive simulaiton gate channel m, 6-6-12 bits, normalised 0..1 . 116
B.13 Simulated NN autoregressive simulaiton gate channel h, 6-6-12 bits, normalised 0..1 . . 117
B.14 Simulated NN autoregressive simulaiton IK, 8-6-12 bits, normalised -16..0 . . . 117
B.15 Simulated NN autoregressive simulation INa, 8-6-12 bits, normalised 0..+16 . . . 118
B.16 Combined LUTs merging Design 3 and 4: gates n,m,h and IK & Ileak are Neural-Network-derived LUTs, while INa is implemented as the simple $g_{Na}m^3$-LUT and 2 DSP blocks . . 118
B.17 Simulated NN autoregressive simulation of $n, m, h, V, I \rightarrow \frac{dV}{dt}$ . . . 119
B.18 Neuronal network simulation using the Design 3 [IONIC-SUM] core for 1250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.1 to -0.4. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference failed neuron (ID=0) (bottom). . . . 120
B.19 Neuronal network simulation using the Design 3 [IONIC-SUM] core for 1250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.1 to -0.4. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference neuron (ID=273) (bottom). . . . 121

List of Tables

2.1 Comparison of platforms for real-time large-scale neural emulation, adapted from Table 2.1 in [34] . . . 12
2.2 FPGA classification and representative resource specifications . . . 14
2.3 FPGA implementations of Hodgkin–Huxley neuron models, showing resource usage (LUTs, DSPs, operating frequency, power) and model configuration (neuron model, timestep Δt, number of neurons, and FPGA device). . . . 19
2.4 Comparison of FPGA Hodgkin–Huxley neuron implementations using normalised per-neuron metrics (LUTs, DSPs, power, GFLOPS), ordered by LUT usage per neuron . . . 19

3.1 Design goals and corresponding design objectives . . . 22
3.2 Design space exploration of FPGA implementations of the Hodgkin–Huxley model, where the designs are additionally visualised in Figure 3.12 . . . 33

5.1 Hodgkin-Huxley simulation parameters used for all simulations . . . 50

7.1 Toolchain used for simulation, synthesis, and optimisation . . . 75
7.2 Memory reduction achieved by the adapted ReducedLUT compression algorithm for the gate-channel LUTs . . . 75
7.3 Generated RTL file sizes before and after LUT compression . . . 75
7.4 Logic Blocks Count of each gate-channel LUT after LUT compression and individual synthesis on an Artix-7 FPGA device with 20,800 Logic Blocks available . . . 76
7.5 Memory reduction achieved after applying delta prediction optimisation and ReducedLUT compression . . . 77
7.6 Generated Verilog file sizes for gate-channel LUTs after applying delta prediction optimisation and ReducedLUT compression . . . 78
7.7 Summarised autoregressive Neural Network prediction statistics per gate-channel. The calculation and interpretation of the MSE, RMSE, and R^2 metrics are described in Section 5.2. Corresponding prediction plots are provided in Section B.4.1. . . . 81
7.8 Generated Verilog sizes for each NeuraLUT-based gate-channel implementation, where all the corresponding perceptron-based LUT sizes are summed together to show the total result . . . 81
7.9 Autoregressive prediction Neural Network metrics for Design 1 [MONOLITH] with NeuraLUT . . . 81
7.10 Summarised autoregressive Neural Network prediction statistics per gate-channel with reduced input and hidden resolution. The corresponding plots can be found in Section B.4.2 . . . 87
7.11 FPGA Resource Utilisation Summary by synthesising the full integrated system corresponding to Design 4[HYBRID] on mid-range (XC7A200T) and a low-end (XC7A35T) FPGA device . . . 88
7.12 Summarised autoregressive Neural Network prediction statistics for the Potassium and Sodium currents with reduced input (8b) and hidden (6b) resolution. The corresponding plots can be found in Section B.4.3. . . . 88
7.13 Generated Verilog sizes and Vivado synthesis logic usage for each ionic current LUT implementation after all optimisations, where all the corresponding perceptron-based LUT sizes are summed together to show the total result. . . . 88
7.14 FPGA Resource Utilisation Summary by synthesising the full integrated system corresponding to Design 3 [IONIC-SUM] on mid-range (XC7A200T) and a low-end (XC7A35T) FPGA device . . . 90

7.15 Summarised autoregressive Neural Network prediction statistics for the 5-input Voltage LUT for design 2 within Optimisation 4. The corresponding plots can be found in Section B.4.4. 91
7.16 Generated Verilog file sizes for the Design 2 voltage-membrane Neural-Network LUT before and after compression 92
7.17 Total LUT size in bits of the Design 2 voltage-membrane Neural-Network LUT before and after compression. 92
7.18 Design-space exploration matrix summarising the results from the optimisation trajectory (1 to 4) on specific design architectures (MONOLITH to HYBRID). The colour scale encodes a qualitative progression from infeasible configurations (red) to near-optimal hardware trade-offs (green) across stability, LUT usage, and DSP cost. 93
7.19 Place-and-route timing and power results for the final neuronal network implementations Design 3 [IONIC-SUM] and Design 4 [HYBRID] on an Artix-7 XC7A200T FPGA device . 97
7.20 Place-and-route timing and power results for the final neuronal network implementations Design 3 [IONIC-SUM] and Design 4 [HYBRID] on an Artix UltraScale+ XCAU10P FPGA device 97
7.21 Performance comparison of the proposed neuronal network design with the Design 4 [HYBRID] neuron core with related work from Table 7.21, using a selected set of comparable metrics 99

8.1 Qualitative final design comparison after optimisations across architectural criteria using star ratings 102

Nomenclature

List of abbreviations

ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuit
BRAM	Block Random-Access Memory
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
DSP	Digital Signal Processor
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HH	Hodgkin-Huxley
HYBRID	Hybrid Architecture
IONIC-SUM	Ionic Current Summation Architecture
LIF	Leaky Integrate-and-Fire
LSB	Least Significant Bit
LUT	Look-Up Table
MLC	Multi-Level Compression
MLP	Multi-Layer Perceptron
MONOLITH	Monolithic Architecture
NCL	Neuro Computing Lab
NN	Neural Network
QAT	Quantisation-Aware Training
RTL	Register-Transfer Level
SPLIT	Split Voltage and Gate Architecture
VHDL	Very High Speed Integrated Circuit Hardware Description Language
CLB	Configurable Logic Block
MSB	Most Significant Bit

List of Symbols

Symbol	Definition	Unit
V_m	Membrane potential	$[mV]$
I_{stim}	Stimulus current	$[\mu A/cm^2]$
C_m	Membrane capacitance	$[\mu F/cm^2]$
$\bar{g}_K$	Maximum Potassium conductance	$[mS/cm^2]$
$\bar{g}_{Na}$	Maximum Sodium conductance	$[mS/cm^2]$
g_l	Constant leak channel conductance	$[mS/cm^2]$
n, m, h	Ion-channel gating state variables	$-$
α, β	Voltage-dependent gating rate functions	$[ms^{-1}]$
Δt	Numerical simulation step size (Euler timestep)	$[ms]$
B	Quantisation bit-width resolution	$[bits]$
S	Absolute total size of flat uncompressed truth-table entries	$-$

1

Introduction

Up to one billion people around the world struggle with brain and nerve disorders that severely affect their every every day lives [1]. Current treatments often focus on managing symptoms and slowing disease progression, while effective methods for restoring damaged neural circuits remain limited [2]. A key emerging direction in neuroprosthetics is the development of neuromorphic twins, which are computational systems designed to emulate and interact with biological neural circuits in real time [3]. Such systems must operate under strict constraints: they must be biologically accurate enough to capture relevant neural dynamics, while also being sufficiently efficient to support real-time operation in embedded hardware [4]. Achieving this balance is essential for enabling closed-loop neuroprosthetic devices that can seamlessly integrate with living neural tissue and restore lost function.

Designing such neuromorphic twins requires neuron models that balance biological plausibility and computational feasibility. While simplified models such as the Leaky-Integrate-and-Fire (LIF) model provide computational efficiency, they abstract away important underlying neuronal behaviour [5–7]. The Hodgkin-Huxley (HH) model provides a more biologically accurate representation by directly modelling the voltage-gated ion channels responsible for neuronal spiking dynamics [8]. While CPU and GPU platforms provide flexible simulation environments, their architectures are not optimised for the computational characteristics and power constraints of embedded HH-based neuroprosthetic applications. FPGAs provide a suitable compromise by combining application-specific hardware structures with reconfigurability, enabling efficient mapping of HH dynamics onto dedicated logic and memory resources [9]. On top of that, they can serve as a prototyping platform for ASIC or chip designs in the following research.

A major challenge in FPGA-based HH implementations is the trade-off between accuracy and the scalability of neuronal network size. The nonlinear dynamics of the Hodgkin-Huxley equations require arithmetic operations, including multiplication, accumulation, and evaluation of nonlinear functions such as exponentials. While the arithmetic components of the differential equations can be efficiently mapped to dedicated Digital Signal Processing (DSP) blocks, nonlinear functions typically require additional approximation structures, such as logic combined with arithmetic resources, to achieve accurate and efficient hardware implementations [10]. As the number of simulated neurons increases, these computational structures must be replicated, causing resource utilisation and power consumption to grow with network size until the available FPGA capacity becomes a limiting factor.

Several FPGA-based HH implementations have demonstrated real-time simulation while maintaining accuracy compared to software simulations. To improve scalability, researchers have explored approximate arithmetic approaches that replace expensive mathematical operations with lower-cost structures. These approaches include piecewise-linear approximations [11, 12], power-of-two-based (CORDIC) approximations [13], lower-order functions [14], and other approximation techniques. Although these methods significantly reduce resource utilisation, they introduce a trade-off between hardware efficiency and numerical accuracy. Another approach is to replace explicit mathematical functions with lookup tables (LUTs), in which nonlinear functions are evaluated through memory accesses rather

than arithmetic operations. This principle has previously been explored in neuron model implementations to reduce computational complexity [15, 16]. These LUT-based approaches shift the design cost from DSP blocks towards logic resources and memory (BRAM). This trade-off can improve resource efficiency when DSP blocks become the limiting factor, enabling larger-scale neuronal network simulations on the same FPGA device. However, it also requires a large memory capacity to maintain sufficient resolution, creating a memory scalability challenge [15].

Recent advances in Neural Network-based function approximators, however, have demonstrated that complex nonlinear mappings can be represented efficiently and transformed into compact LUT structures using tools from NeuraLUT and Reducedlut [17, 18]. NeuraLUT leverages neural networks' ability to learn complex nonlinear relationships by combining many simple transformations into a compact model. After training, the learned network behaviour can be converted into LUT structures, enabling the approximation to be implemented directly in hardware without performing neural network computations at runtime. ReducedLUT further reduces the resulting LUT size by applying compression and optimisation techniques while maintaining the represented function. Since prior research has shown that artificial neural networks can successfully learn the Hodgkin-Huxley dynamics [19], applying NeuraLUT to these dynamics enables the possibility of replacing individual mathematical operations with learned representations while preserving the original biological behaviour.

Therefore, although FPGA-based implementations provide a promising platform for biologically realistic Hodgkin-Huxley simulation, existing approaches remain limited by a trade-off among accuracy, hardware resource efficiency, and scalability to large neuronal networks.

1.1. Motivation

While FPGA-based implementations have demonstrated real-time multi-neuron simulation capabilities, achieving high neuron density within a given FPGA resource budget remains challenging under strict DSP and memory constraints due to the high cost of nonlinear arithmetic operations. To address this limitation, this thesis investigates a Neural Network-based lookup table (NeuraLUT) representation of the HH dynamics, aiming to achieve more resource-efficient and scalable hardware mapping while preserving biologically relevant behaviour.

The expected outcomes of this approach are:

- **Reduced dependency on DSP-intensive arithmetic operations:** By replacing computationally expensive arithmetic operations with lookup tables, the dependency on DSP resources is expected to be reduced, potentially enabling Hodgkin-Huxley simulation on resource-constrained FPGA platforms where insufficient DSP availability would otherwise prevent a hardware implementation.
- **Improved scalability under hardware resource constraints:** The compact footprint of LUT-based neuron representations is expected to enable a time-multiplexed architecture, supporting a higher-density design in which a single shared neuron core can serve multiple neuron instances, enabling large-scale neuronal network simulation within a fixed FPGA resource budget.
- **Biologically grounded and potentially extensible representation:** By training Neural Network models on HH-generated data, the learned representations are expected to reproduce the dynamics of the original neuron model. Furthermore, decomposing ion-channel computations into independent function blocks may enable the incorporation of additional ionic channels by training and integrating new channel representations, potentially avoiding redesign of the underlying hardware datapath.
- **Configurable accuracy-resource trade-off through neural network parameterisation:** Unlike fixed analytical approximations, the NN-based LUT generation exposes a controllable trade-off between biological accuracy and hardware resource utilisation, by adjusting network hyperparameters and quantisation precision.

Consequently, this thesis develops and evaluates a neural network-based LUT approach for accurate, resource-efficient, and scalable Hodgkin-Huxley implementations on low-resource FPGAs, providing a potential foundation for future neuromorphic platforms and closed-loop neuroprosthetic hardware systems.

1.2. Research Question and Thesis Goals

Biophysically detailed neuron models, such as the conductance-based HH model, provide accurate representations of neuronal dynamics but, in their original form, are computationally expensive due to their nonlinearity. For scaling real-time simulation to large neuronal networks on resource-constrained FPGA platforms, the computational cost of evaluating these dynamics becomes a limiting factor.

Therefore, the research question of this thesis can be formulated as follows:

How can Neural Network-based LUT representations increase the achievable Hodgkin–Huxley neuron simulation density on FPGA architectures under a fixed resource budget while preserving physiologically realistic multi-neuron dynamics?

Based on this research question, the following goals are defined for this thesis:

1. Characterisation of computational bottlenecks in FPGA-based Hodgkin-Huxley hardware simulation
2. Exploration of FPGA architectures with different granularity levels for decomposing Hodgkin-Huxley computations with respect to resource efficiency and neuron density
3. Investigation of the biological accuracy of a NeuraLUT-based representation of the nonlinear dynamics of a Hodgkin-Huxley model
4. Implementation of a resource-constrained FPGA architecture for a multi-neuron simulation
5. Evaluation of scalability in terms of neuron density, resource-efficiency, and accuracy trade-offs in the designed system

1.3. Thesis Methodology

To understand the challenges and design opportunities of real-time brain simulation on FPGAs under a fixed resource budget, the problem and its scope are first explained. Then, some relevant background, including the tools used for LUT-based generation and compression, is briefly described. After comparing this work with related work and evaluating it against relevant performance metrics, a list of specific design goals is identified.

With the clear hardware constraints and design goals set, different possible design architectures are proposed that vary in the granularity of the HH-dynamics decomposition. Afterwards, the optimisations required to meet the constraints are explained, showing exactly which design goals they address and the impacts on relevant designs. Hereafter, the three simulation structures are outlined and summarised, including the entire neural network training approach, the LUT compression algorithm and the neuronal network topology.

Finally, after evaluating the designs and optimisations, the final designs are implemented in a multi-neuron network simulation and tested on a target device. The resulting network is benchmarked on the previously mentioned metrics and compared with state-of-the-art related work.

1.4. Thesis Organisation

In Chapter 2, background information and relevant topics are presented, followed by the problem formulation and comparison with related works. In Chapter 3, the details of the proposed designs for decomposing the HH model are discussed in depth, going from coarse-grained to fine-grained architectures. In Chapter 4, the implemented design optimisations are thoroughly explained and evaluated on the chosen neuron core metrics. In Chapter 5, the simulation process is outlined. Additionally, the required tools for the optimisations, such as NeuraLUT and ReducedLUT, are explained in more detail to provide a thorough understanding of their challenges and shortcomings. In the same chapter, the applied adaptations to the tools to realise the desired outcomes are also presented. In Chapter 6, the first proof of concept of a multi-neuron network is introduced, where the neuron-core wrapper that time-multiplexes the neuron core and the full synapse controller is thoroughly explained. In Chapter 7, the performance results of the designs after each optimisation are presented, whereafter the neuronal networks based on these designs are evaluated and compared with related work. Finally, in Chapter 8, concluding remarks and suggested future research advice are presented.

2

Background & Related Work

2.1. The Neuron

The human brain consists of billions of neurons that communicate through electrical and chemical signals. Although neurons exhibit highly complex biological behaviour, their activity can be described through changes in the membrane potential caused by the movement of ions across the cell membrane. The separation of charged ions between the inside and outside of the neuron creates a voltage difference, which dynamically changes as ions flow through selective openings in the membrane, known as gate channels. Consequently, neuronal behaviour can be represented as a dynamic electrical system in which ionic currents determine the evolution of the membrane voltage.

A key mechanism enabling communication between neurons is the generation and propagation of action potentials. When a sufficient amount of accumulated stimulus current is reached, voltage-dependent gate channels open and allow specific ions to flow into or out of the cell. This temporarily changes the membrane voltage, pulling it toward positive and pushing it back into negative ranges. These action potentials travel across the membrane and along the connecting pathways, called axons, to neighbouring neurons, thereby inducing a stimulus current that activates the neuron. This process is illustrated in Figure 2.1.

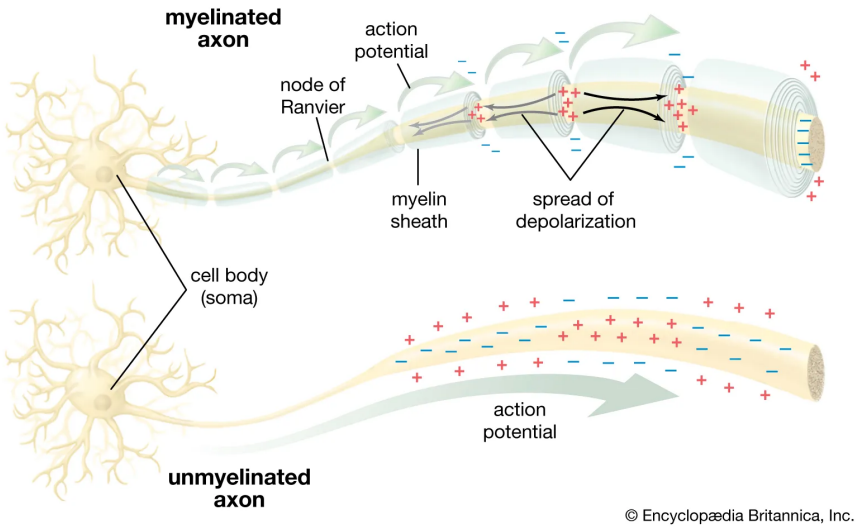


Figure 2.1: Propagation of action potentials in axons, adapted from [20]

The resting membrane potential is maintained by the steady-state ion-concentration difference across the neuronal membrane. In particular, sodium (Na^+) ions are predominantly present outside the neu-

ron, while potassium (K^+) ions are concentrated inside the cell, as illustrated in Figure 2.2. This concentration imbalance creates a negative membrane potential, typically around $-65mV$ at rest.

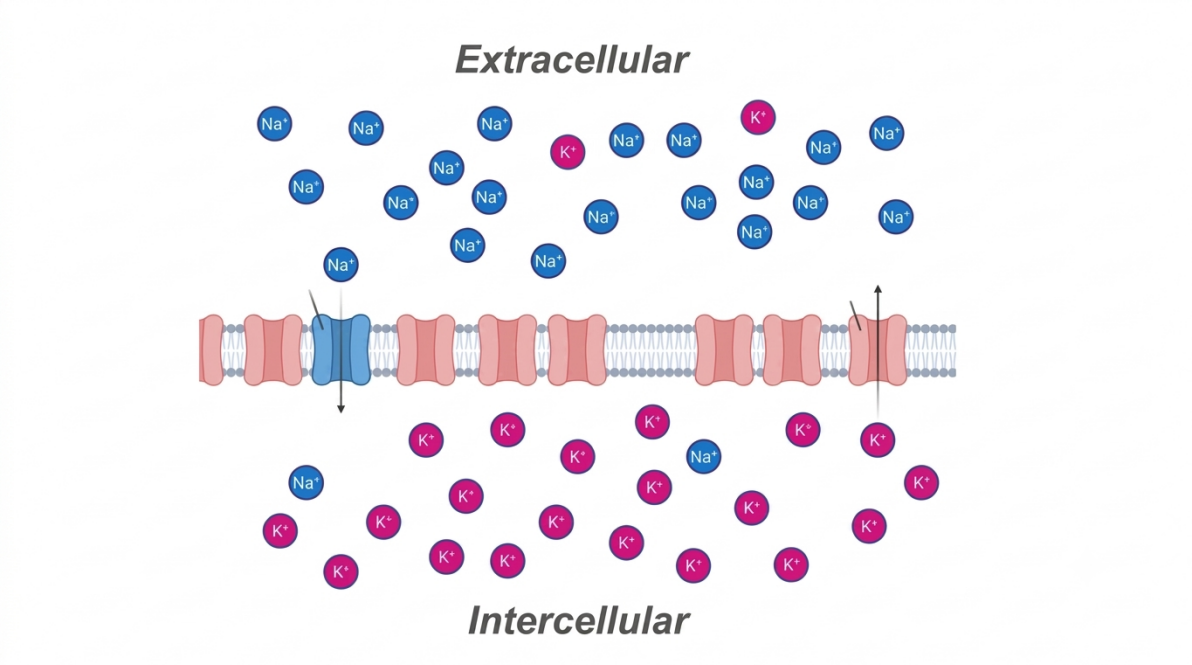
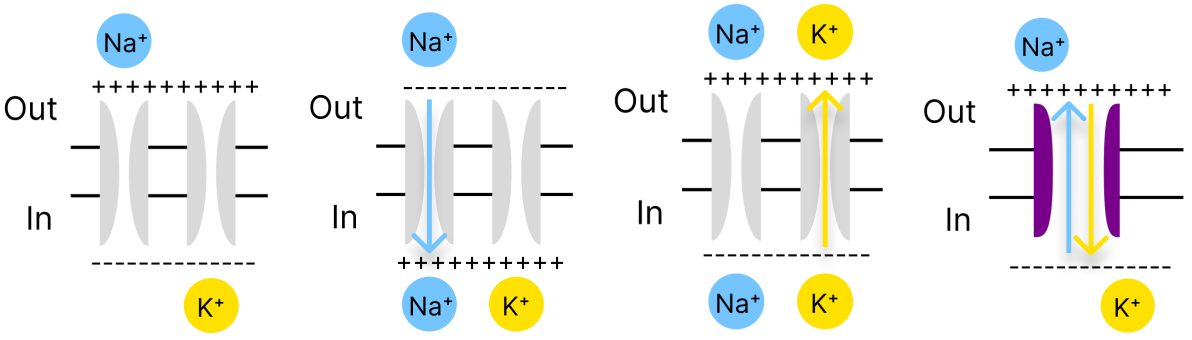


Figure 2.2: Illustration of ion concentration inside and outside the neuron membrane in the resting state, adapted from the illustration created by Jim Hutchins and Alexa Crookston with BioRender [21]

During an action potential, gate channels regulate the movement of ions through the membrane. This is done with a characteristic process illustrated in Figure 2.3. Initially, sodium channels open, allowing (Na^+) ions to enter the cell and cause the first phase of the action potential: the depolarisation. In this phase, the inside of the cell becomes largely positively charged, pulling the membrane voltage to $+50mV$, as shown in Figure 2.4. Following, potassium channels open, allowing (K^+) ions to leave the cell, resulting in the repolarisation phase, in which the membrane voltage descends again. The delayed closing of potassium channels temporarily reduces the membrane voltage to slightly below the resting potential, towards the $-80mV$ range, which is characterised as the hyperpolarisation phase. Finally, the ion concentrations are slowly restored through leaking channels, which resets the membrane voltage to its resting potential.



(a) Resting phase **(b)** Depolarisation phase **(c)** Repolarisation phase **(d)** Hyperpolarisation phase

Figure 2.3: Illustration of ion flow across the neuron membrane for the four different phases of an action potential

The full corresponding action potential due to these changes in membrane voltage per phase is illustrated in Figure 2.4.

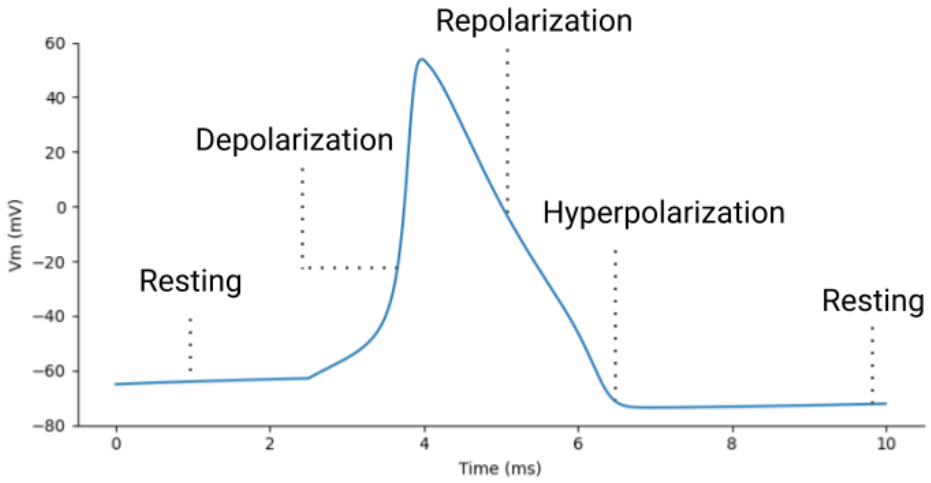


Figure 2.4: Illustration of a neuronal action potential with annotations for each changing phase upon arrival of a stimulus signal

2.1.1. Computational Neuron Models

The electrical behaviour of biological neurons results from complex interactions between ion channels and intracellular dynamics. To enable computational analysis and simulation, these biological processes are represented using mathematical neuron models. Such models differ in their level of abstraction, creating a trade-off between biological plausibility and computational efficiency. Simplified models reduce neuronal dynamics to enable large-scale simulations, whereas detailed models aim to reproduce the underlying mechanisms responsible for neuronal behaviour.

The Leaky Integrate-and-Fire (LIF) model is one of the most widely used simplified neuron models due to its low computational complexity and suitability for large-scale artificial Spiking Neural Network (SNN) simulations [7]. The model represents a neuron as a membrane-potential accumulator with a leaky term and generates a spike when the membrane potential crosses a threshold. However, this abstraction omits important biological processes, including the gate channel dynamics responsible for action potential generation [5, 6]. Therefore, although LIF models are effective for investigating network-level behaviour, they cannot reproduce the intrinsic biophysical neuronal dynamics.

To bridge the gap between computationally efficient simplified models and biologically realistic models, phenomenological neuron models have been developed. These models reproduce input-output behaviour and diverse firing patterns while avoiding detailed biophysical mechanisms. Examples include the Izhikevich model [22] and the Adaptive Exponential Integrate-and-Fire (AdEx) model [23], which introduce additional mechanisms, such as an exponential rate of change of the membrane voltage, thereby enabling a wider range of neuronal behaviour to be replicated. These approaches provide a satisfactory balance between computational complexity and biological plausibility for large-scale networks where spiking behaviour is the dominant focus rather than exact biological mechanisms.
Figure 2.5 compares several neuron models based on their computational cost and biological plausibility [22]. However, the figure's biological plausibility on the y-axis refers to the ability to reproduce a variety of neuronal firing patterns and spiking behaviours, such as phasic spiking and tonic bursting [22]. It does not represent the extent to which the models reproduce underlying biophysical dynamics. For applications requiring biophysically realistic models, such as those targeting digital counterparts of biological neurons, conductance-based models are necessary. These models explicitly represent the physical mechanisms that drive the evolution of the membrane voltage through ion channel dynamics.

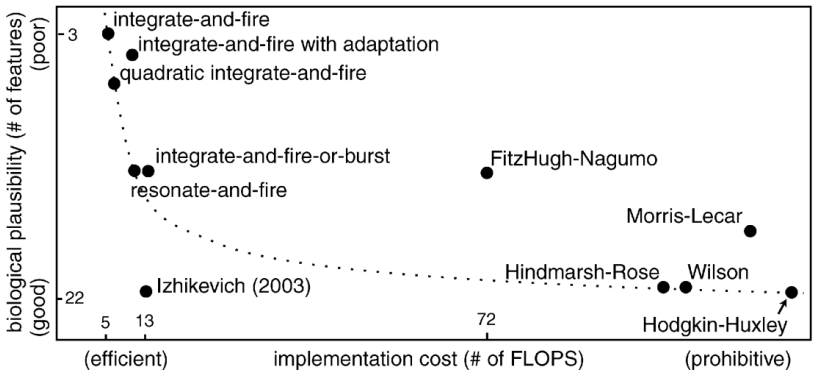


Figure 2.5: Overview of different neuron models with respect to computational complexity (x-axis) and level of biological plausibility (y-axis), interpreted here in terms of spiking behaviour, adapted from [22]

The Hodgkin-Huxley (HH) model provides one of the most established representations of neuronal dynamics by explicitly modelling ion gate channels that drive the voltage transitions and spike generation [8]. Although introduced in 1952, the model remains highly relevant due to its ability to capture actual biological action potentials and its continued use in computational neuroscience research [24].

Hodgkin-Huxley Model

The Hodgkin-Huxley model was introduced by Alan Hodgkin and Andrew Huxley in 1952 to mathematically describe the electrical behaviour of neurons. The model represents the membrane as a capacitive electrical circuit in which ion gate channels are represented as variable conductances. These conductances depend on the activation and inactivation state of the biological gate channels, allowing the model to reproduce the exact action potential phases during stimulus. As illustrated in Figure 2.6, the classical HH model describes three voltage-dependent gating variables associated with the sodium and potassium channels, together with a passive leakage channel without an associated dynamic variable:

- m gate: Controls the sodium Na^+ channel activation. Opening this gate allows sodium ions to flow into the cell, directly causing depolarisation.
- h gate: Controls the sodium Na^+ channel inactivation. From its resting open state, this gate slowly closes the channel, thereby stopping the inflow of sodium ions and contributing to the repolarisation phase.
- n gate: Controls the potassium K^+ channel activation. The late opening of this gate allows potassium to flow out of the cell, causing the repolarisation and hyperpolarisation phases of the membrane.
- leak channel: This channel represents the membrane's passive permeability to ions, contributing to the stabilisation of the resting membrane potential through a continuous background ion flow.

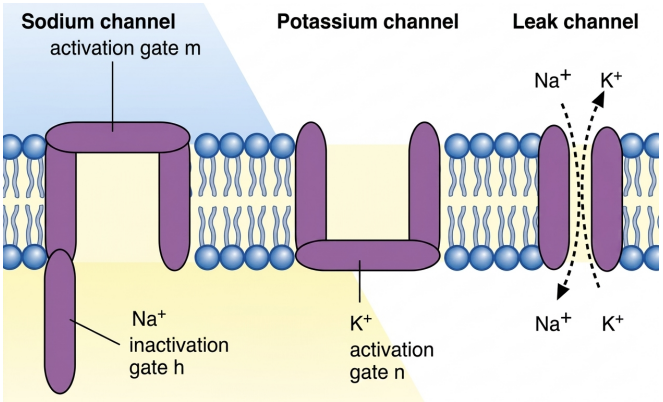


Figure 2.6: Illustration of ion channels for sodium and potassium flow with their associated gates n,m and h, and the leak channel, adapted from [25]

Figure 2.7 shows the temporal changes of the gating variables in relation to the resulting action potential.

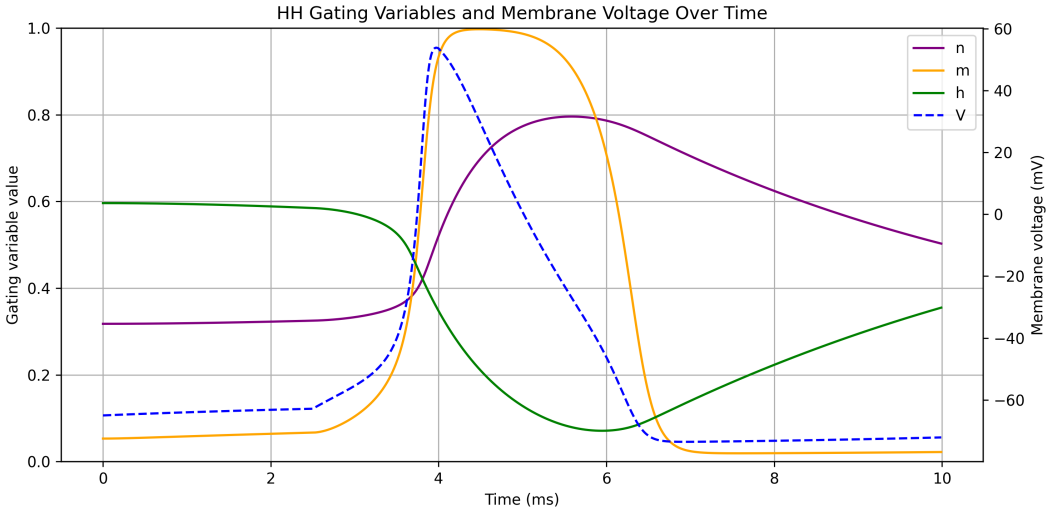


Figure 2.7: Illustration of the temporal change of the ion gate channels n,m and h with respect to the action potential (V_m)

The corresponding electrical representation of the HH model is shown in Figure 2.8. In this representation, the neuronal membrane is modelled as a capacitor, reflecting the cell membrane's ability to store charge. This is the analogy for the imbalance in ionic concentrations between inside and outside of the cell, as was illustrated in Figure 2.2. Therefore, the membrane voltage V_m is represented as the voltage across this capacitor. The ionic channels represent the flow of charged particles as parallel ionic currents induced by voltage sources V_{Na}, V_K, V_{leak}, also known as reversal potentials, going through the gate-dependent variable conductances gNa and g_K, and the constant conductance g_{leak}.

An important detail in the circuit representation is the sign convention of the reversal potentials. Since the physical movement of positively charged ions and the conventional electrical current direction are opposite, the sodium current I_{Na} flows opposite to the inflow of sodium ions (Na^+) during depolarisation. Therefore, the V_{Na} source is drawn with reversed polarity in the circuit representation. This ensures the resulting sodium current follows the consistent mathematical sign convention used for the ionic currents and can be combined directly with I_K and I_{leak} in the membrane voltage equation according to Kirchhoff's current law.

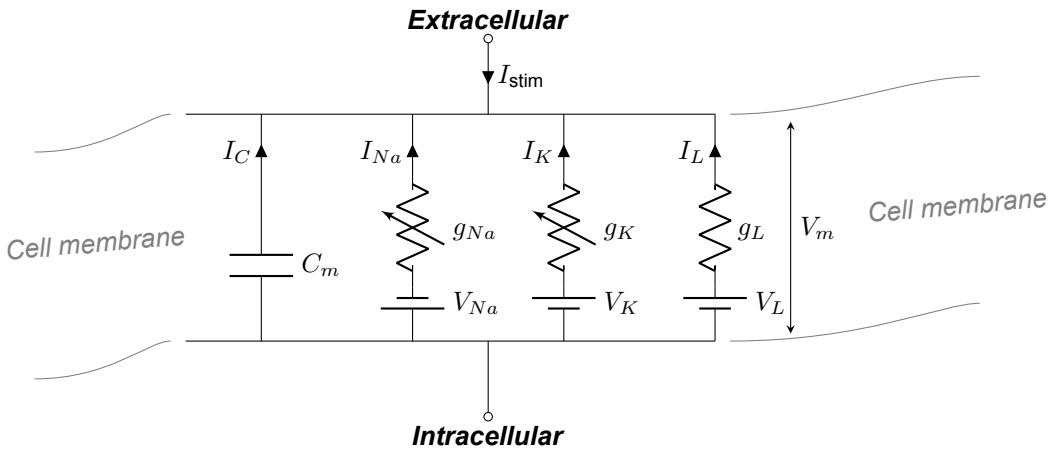


Figure 2.8: Electrical circuit representation of the Hodgkin-Huxley model, adapted from [26]

The net inflow must equal the total outflow from the circuit according to Kirchhoff's Law, so the incoming stimulus is equal to the sum of the electrical representation of the capacitance via $I_C = C_m \frac{dV}{dt}$ with the separate ionic currents I_{Na}, I_K, I_{leak}. The HH model defines the variable conductances in the circuit from Figure 2.8 as constants multiplied by the relevant gate variables, where the constants $\bar{g}_{Na}$ and $\bar{g}_K$, as well as g_{leak}, represent the maximum value of conductance. The full resulting equation is displayed by Equation (2.1).

$$\begin{aligned} I_{stim} &= C_m \frac{dV_m}{dt} + g_K(V_m - V_K) + g_{Na}(V_m - V_{Na}) + g_l(V_m - V_l) \\ &= C_m \frac{dV_m}{dt} + \bar{g}_K n^4 (V_m - V_K) + \bar{g}_{Na} m^3 h (V_m - V_{Na}) + g_l(V_m - V_l) \end{aligned} \tag{2.1}$$

The gating variables n, m, and h, are defined by their derivative equations (2.2)–(2.4).

$$\frac{dn}{dt} = \alpha_n(V_m)(1-n) - \beta_n(V_m)n \tag{2.2}$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1-m) - \beta_m(V_m)m \tag{2.3}$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1-h) - \beta_h(V_m)h \tag{2.4}$$

In this formulation, each variable represents the probability of a specific ion-channel state being open, and its dynamics are driven by voltage-dependent transitions between closed and open configurations. These transitions are captured by the rate functions $\alpha_x(V_m)$ and $\beta_x(V_m)$, which determine how rapidly each gating variable responds to changes in membrane potential. In the classical Hodgkin–Huxley formulation, they take the following form:

$$\alpha_n(V_m) = \frac{0.01(V_m + 55)}{1 - \exp(-(V_m + 55)/10)} \qquad \beta_n(V_m) = 0.125 \exp(-(V_m + 65)/80) \tag{2.5}$$

$$\alpha_m(V_m) = \frac{0.1(V_m + 40)}{1 - \exp(-(V_m + 40)/10)} \qquad \beta_m(V_m) = 4 \exp(-(V_m + 65)/18) \tag{2.6}$$

$$\alpha_h(V_m) = 0.07 \exp(-(V_m + 65)/20) \qquad \beta_h(V_m) = \frac{1}{1 + \exp(-(V_m + 35)/10)} \tag{2.7}$$

Therefore, the HH model consists of coupled, nonlinear, differential equations in which changes in membrane voltage influence the gating variables, and vice versa. The classical HH model represents only a subset of the biological processes present in real neurons. Depending on the neuron type and brain region, additional ionic mechanisms, such as calcium dynamics, can be incorporated. However, the classical HH equation remains a widely used foundation for studying neural behaviour due to its balance between biological accuracy and model complexity.

2.1.2. Numerical simulation of HH dynamics

Since the HH model consists of a set of coupled differential equations, numerical integration is required to simulate its behaviour over discrete time steps. The integration timestep determines how accurately the neuron's continuous dynamics are represented. In particular, the rapid changes induced by gate channels require sufficiently small time steps to capture realistic membrane dynamics. However, reducing the timestep increases the number of required updates, thereby increasing the computational cost of HH simulation. A commonly used method for numerically solving differential equations is the forward Euler method. Instead of continuously computing the evolution of a variable, the method approximates its change over a small time step.

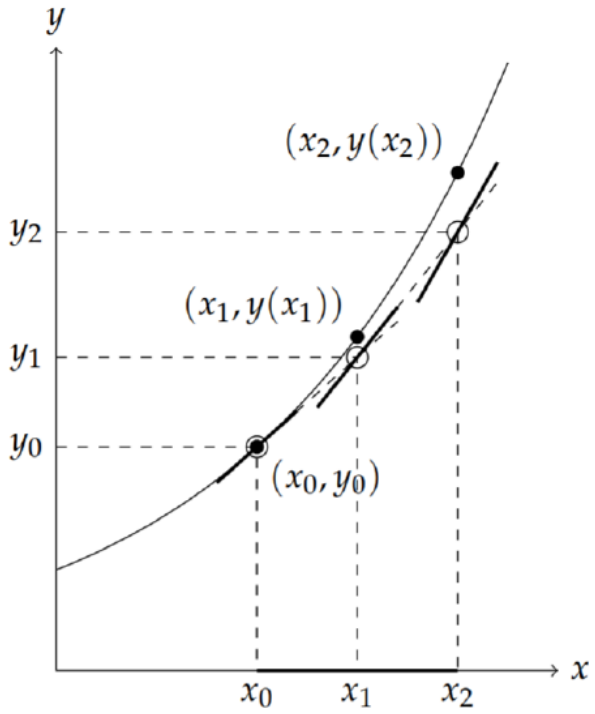


Figure 2.9: Illustration of the numerical solution of a differential equation using the forward Euler method, adapted from Figure 10.1.1 in [27]

The principle of this approximation is illustrated in Figure 2.9, where the slope at a known point (x_0, y_0) is used to estimate the next value y_1 at x_1. By repeatedly applying this update, the continuous function can be approximated as a sequence of discrete states. Concretely, for a general differential equation:

$$\frac{dy}{dx} = f(x, y), \tag{2.8}$$

the derivative describes the rate of change of the variable y at a specific point in time. Over a small timestep Δx, the change in the state variable can be approximated as:

$$\Delta y \approx \frac{dy}{dx}\Delta t \tag{2.9}$$

Since the change in the state variable is defined as

$$\Delta y = y_{n+1} - y_n, \tag{2.10}$$

the next state can be estimated as:

$$y_{n+1} = y_n + f(x_n, y_n)\Delta x \tag{2.11}$$

where y_n represents the current state and y_{n+1} the estimated state after one timestep. The forward Euler method, therefore, converts the continuous differential equation into a sequence of discrete updates.

The same principle may be applied to the HH model, in which the membrane voltage and gating variables are updated at each time step using their respective derivatives. Due to the nonlinear nature of the HH equations, small time steps are required to maintain biologically realistic behaviour, making numerical integration a significant contributor to the computational complexity of HH-based simulations.

2.1.3. Neuronal Network

While the Hodgkin-Huxley model describes the dynamics of an individual neuron, practical applications such as large-scale brain simulation and neuroprosthetic systems require the interaction of multiple connected neurons. Therefore, to evaluate whether a neuron implementation can serve as a scalable hardware block, the design must be extended from a single neuron core towards a neuronal network in which neurons exchange information through synaptic connections.

A commonly used benchmark model for studying neuronal networks is the balanced excitatory-inhibitory network introduced by Brunel [28]. This network consists of populations of excitatory neurons (80%) and inhibitory neurons (20%). Excitatory neurons increase the activity of connected neurons by sending positive stimulating inputs through synapses, while inhibitory neurons suppress network activity by sending negative inputs. The balance between those two populations creates stable network dynamics, preventing both network inactivity, in which neurons fail to receive sufficient stimulation to reach the action potential threshold, and excessive synchronisation, in which all neurons spike simultaneously and rapidly.

The neurons are connected with 10% sparsity, meaning each neuron communicates with only 10% of the total neuron population. Additionally, externally generated Poisson-distributed spike inputs are commonly used to represent background activity from surrounding brain regions. The resulting network structure is illustrated in Figure 2.10.

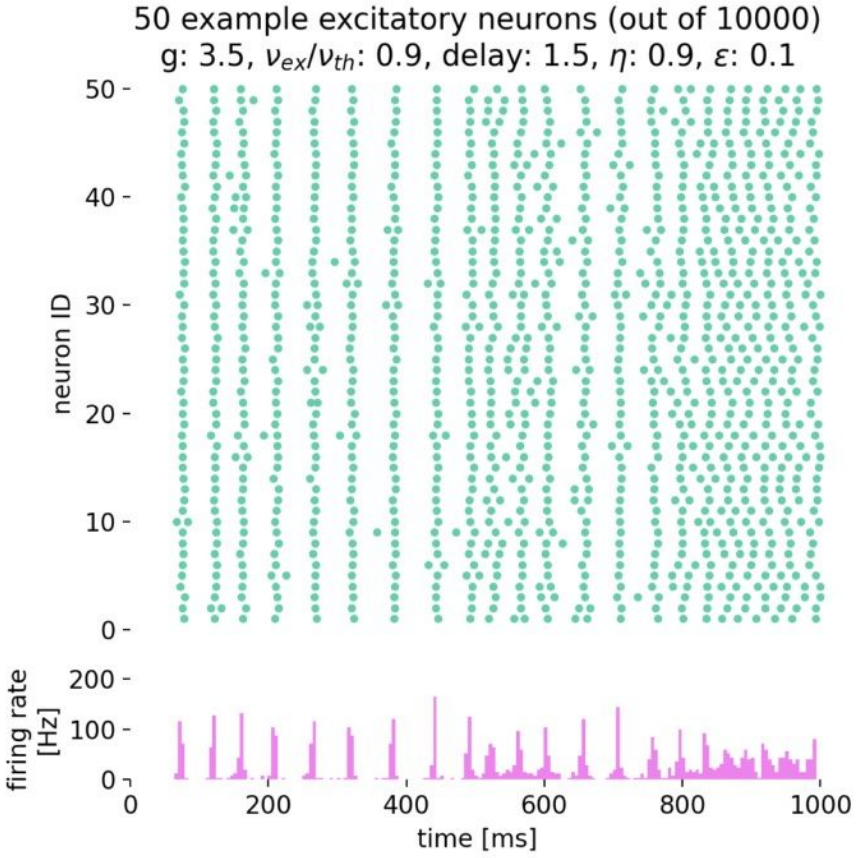


Figure 2.10: Schematic representation of the balanced excitatory-inhibitory network introduced by Brunel [28].

In this thesis, the Brunel network is used as a proof-of-concept to evaluate the scalability of the proposed FPGA neuron architecture. The focus is not on reproducing a complete biological network but on demonstrating the hardware requirements for scaling the neuron core to multi-neuron applications, including synaptic storage, spike scheduling, and inter-neuron communication.

2.2. Brain Simulation on Hardware Platforms

The computational complexity of the Hodgkin-Huxley model makes its implementation challenging for conventional computing platforms. CPUs provide high algorithmic flexibility and enable software-based optimisation of large-scale networks [29], but their von Neumann architecture introduces substantial data-movement overhead due to the physical separation of memory and compute units. GPUs provide high parallel computational throughput for accelerating neuron simulations [30, 31], but their high power consumption makes them less suitable for resource-constrained biomedical systems [32]. Beyond computational efficiency and power constraints, the physical form factor of CPU- and GPU-based systems further limits their applicability in wearable or implantable neuromorphic platforms, where size, integration, and on-body deployment constraints are critical design considerations.

Custom hardware platforms provide an alternative by allowing the computation to be mapped directly onto application-specific architectures. ASICs provide high efficiency and compactness, but their fixed hardware limits post-fabrication adaptability. FPGAs provide a compromise between efficiency and reconfigurability by enabling dedicated hardware structures while remaining reconfigurable. Recent FPGA-based HH implementations have demonstrated the potential of this approach in achieving high simulation performance and energy efficiency [9]. However, the achievable scalability strongly depends on the efficient utilisation of the available FPGA resources. A concise comparison between the four mentioned simulation platforms, CPU, GPU, FPGA and ASIC, is described in Table 2.1 and visualised in an adapted figure from Isik [33] comparing the four platforms for Spiking Neural Network applications (SNN).

Platform	Flexibility	Energy Efficiency	Latency	Development cost
CPU	Very high	Low	Moderate	Low
GPU	High	Low	Moderate	Low
FPGA	Moderate	High	Low	Moderate
ASIC	Very Low	Very high	Very low	Very high

Table 2.1: Comparison of platforms for real-time large-scale neural emulation, adapted from Table 2.1 in [34]

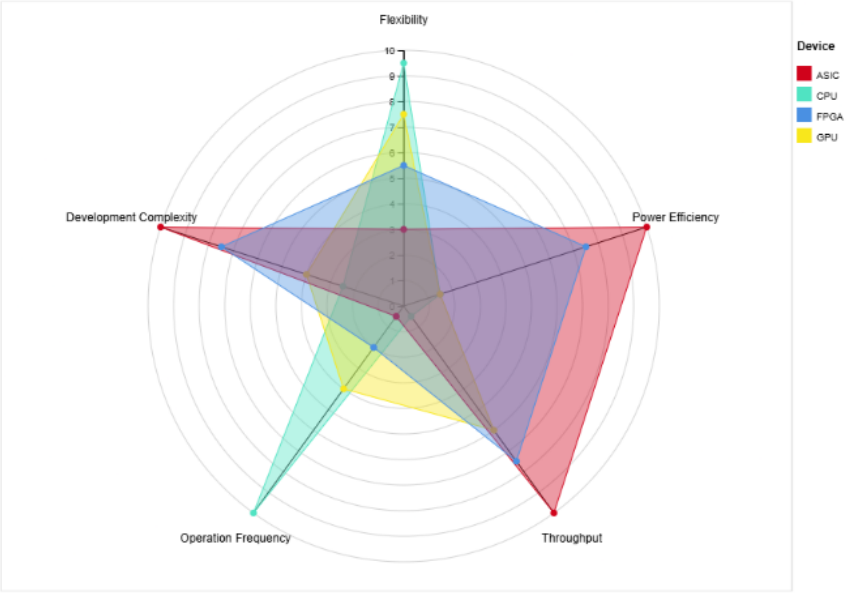


Figure 2.11: Comparison between four platforms, CPU-GPU-FPGA-ASIC, for Spiking Neural Network (SNN) applications, adapted from [33]

2.2.1. FPGA Architecture

An FPGA consists of configurable hardware resources that can be interconnected to implement custom digital circuits. Unlike CPUs and GPUs, where computations are executed sequentially or through

predefined processing units, FPGA designs map operations directly onto hardware structures, enabling high levels of parallelism and deterministic execution.

The main resources relevant for HH implementations are multiplier (DSP) blocks, memory (BRAM) blocks, and logic (LUT) blocks, as shown in Figure 2.12.

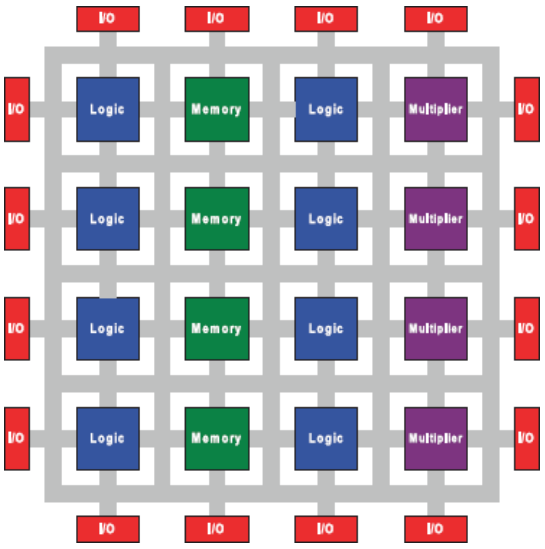


Figure 2.12: Basic FPGA structure, reproduced from Fig.1.1 in [35].

Multiplier blocks consist of DSPs (Digital Signal Processing), which are dedicated arithmetic units optimised for operations such as multiplication and accumulation, making them suitable for implementing the floating-point or fixed-point arithmetic required by HH equations. However, their limited availability becomes a bottleneck when scaling towards large neuronal networks [36]. Furthermore, complex arithmetic operations introduce deeper computational datapaths, which can increase latency and require additional pipelining to maintain high operating frequencies.

Memory blocks, typically implemented via BRAMs (Block Random Access Memory), provide dedicated on-chip memory for storing parameters, intermediate values, or large lookup tables. While BRAM offers efficient and scalable memory storage, it typically introduces a registered read stage (often one to two clock cycles), which can make it slower than distributed LUT-based implementations for small and latency-critical accesses. However, the available BRAM capacity is significantly larger than that of size-limited LUT-based storage.

As shown in Figure 2.13, logic blocks consist of logic elements combined with Lookup Tables (LUTs), which can implement arbitrary combinational functions. By combining multiple LUTs, complex mathematical operations can be approximated or replaced by direct hardware mappings. This makes LUTs particularly interesting for reducing computational complexity in nonlinear functions, which form the main challenge of HH implementations.

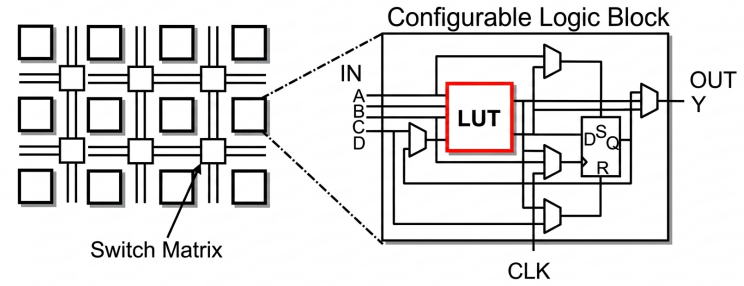


Figure 2.13: Overview of a Configurable Logic Block, adapted from Figure 1a in [37].

Consequently, FPGA-based HH simulation requires balancing the utilisation of DSP blocks, BRAM resources, and Logic blocks to achieve accurate, efficient, and scalable neuronal network simulation.

Look-Up Tables

In FPGA terminology, a LUT refers to a configurable logic element that implements a Boolean function. For example, a 6-LUT can represent any function of six binary inputs by storing the corresponding outputs in a truth table with $2^6 = 64$ possible input combinations. Larger functions require multiple LUTs connected together.

For numerical applications, LUTs can be used to approximate mathematical functions by discretising the input range and storing the corresponding output values. An example of this is shown in Figure 2.14. The required number of LUT resources depends on the input resolution, output precision, and complexity of the represented function. Therefore, directly implementing high-resolution nonlinear functions can require a large number of FPGA resources, motivating approaches that optimise or compress LUT representations. For the remainder of this thesis, the term *logic block* refers to a physical 6-input LUT primitive on the FPGA. The term *LUT* is used to refer to the abstract quantisation of a complex function, which may, in principle, span a very large input domain when considered independently of hardware constraints.

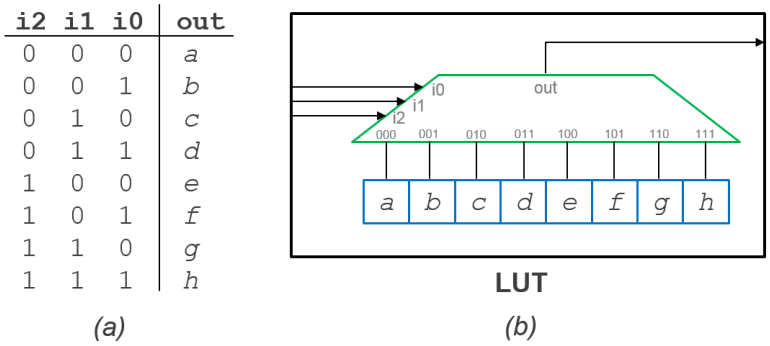


Figure 2.14: Internal structure and operation of a lookup table (LUT), adapted from the RapidWright documentation [38].

FPGA Resource Scale

FPGA devices are available across a wide range of resource capacities, from small, low-power devices to high-performance, accelerator-class devices. Since this thesis focuses on resource-constrained FPGA implementations, the available DSP, BRAM, and LUT resources strongly influence achievable neuron capacity and architectural choices. Table Table 2.2 provides an approximate overview based on representative FPGA families from AMD- and Intel-device specifications [39–42].

	FPGA classification		
Resource	**Low-end**	**Mid-end**	**High-end**
Logic blocks (6-LUT)	10k - 100k	100k - 1M	1M - 9M
DSP blocks	10 - 200	200 - 3k	2k - 6k
BRAM capacity	180 Kb - 6.8 Mb	10 Mb - 50 Mb	50 Mb - 500 Mb

Table 2.2: FPGA classification and representative resource specifications

2.3. Neural Network-based LUT Compression

2.3.1. Neural Networks

Artificial Neural Networks (ANN) are machine learning models that approximate complex relationships between inputs and outputs by learning a set of parameters from example data. Instead of explicitly defining a mathematical expression for a function, neural networks learn an internal representation of this relationship through interconnected relations between computational units called perceptrons. In literature, these units are more commonly referred to as neurons; however, in order not to confuse

them with the biological neurons that this thesis focuses on, the remainder of this thesis will stick to the naming convention of perceptrons for these artificial neurons in neural networks. The network parameters learned during training encode the input-output relations through the combined contribution of all perceptrons in the network. After training, these parameters allow the network to approximate the original function without requiring the original training data or the original mathematical expression. Therefore, a neural network can serve as a compact approximation of a complex nonlinear function, replacing the need for large lookup tables or computationally expensive function evaluations.

Perceptron

The perceptron is a mathematical unit that transforms multiple inputs into a single output value by summing weighted inputs and applying a nonlinear *activation* function. As illustrated in Figure 2.15, each input is multiplied by a corresponding weight, and a bias term is added before applying the activation function. This operation can be expressed as:

$$y = \mathsf{f}\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.12}$$

where x_i represents the input values, w_i the corresponding weights, b the bias term, and $\mathsf{f}()$ the activation function. The weights determine the contribution of each input to the output, while the bias provides an adjustable offset. The activation function introduces nonlinearity, allowing perceptrons to represent relationships beyond simple linear mappings.

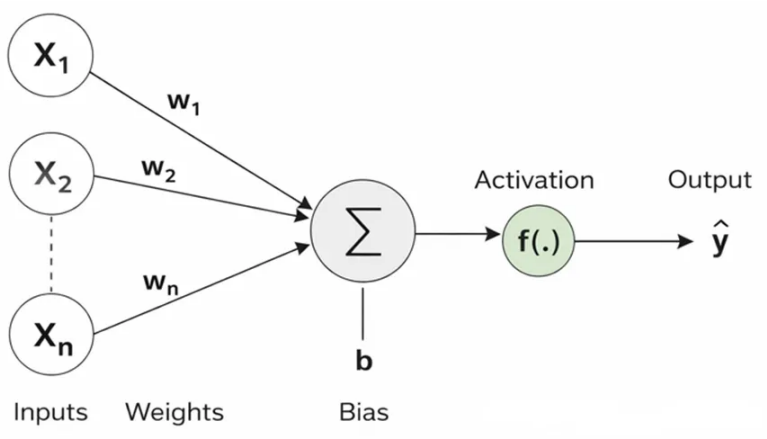


Figure 2.15: Detailed illustration of a perceptron in an artificial neural network, reproduced from [43].

Multi-Layer Perceptrons (MLP)

A single perceptron can only represent limited relationships between inputs and outputs. By combining multiple perceptrons into a layered structure, more complex functions can be approximated. A Multi-Layer Perceptron (MLP) is a feed-forward neural network consisting of an input layer, one or more hidden layers, and an output layer, as illustrated in Figure 2.16. The hidden layers allow the network to learn intermediate feature representations of the input data. Increasing the number of layers or perceptrons increases the representational capability of the network.

The parameters of an MLP consist of the weights and biases of all perceptrons, which are refined iteratively with an *optimiser* algorithm. These parameters are obtained during training, where the predicted and target outputs are compared using a *loss* function. The loss value represents the difference between the predicted and expected output, and is used to determine how the parameters should be adjusted. Through the *backpropagation* process, the gradient of the loss $\frac{\partial L}{\partial \theta}$ with respect to the weight and bias is calculated. L represents the loss function and θ represents the trainable network parameters. An optimisation algorithm, such as *gradient descent*, then updates the parameters in the direction that reduces the loss. This process is repeated over the training dataset until the network converges to a suitable approximation of the desired function with minimised loss.

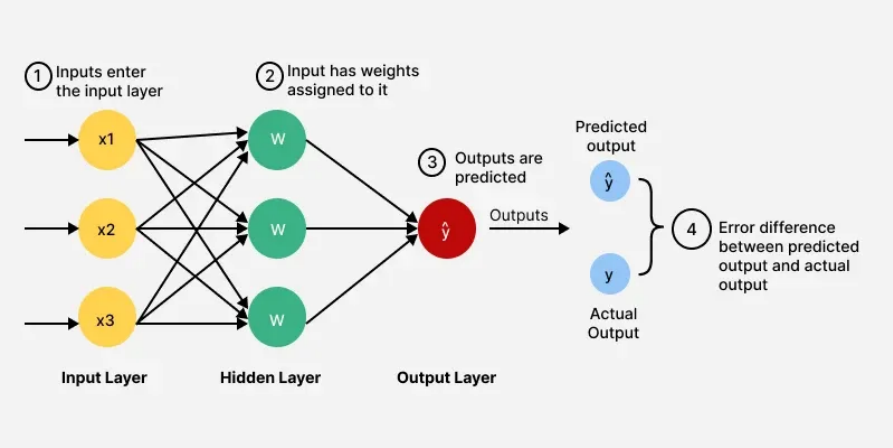


Figure 2.16: Detailed illustration of a feed-forward neural network with 1 hidden layer, reproduced from [44].

2.3.2. NeuraLUT

Neural networks provide a compact representation of complex nonlinear functions through learned parameters. However, directly integrating neural networks on FPGAs often requires multiplication, accumulation, and activation operations, which typically rely on many dedicated arithmetic resources. NeuraLUT, introduced by Constantinides and Andronic [17], proposes an alternative approach that converts neural networks into synthesizable Boolean lookup tables. Instead of implementing the neural network on an FPGA and having it evaluate inputs, we train the network beforehand and convert the input-output relationships into lookup tables, referred to as L-LUTs (Logical LUTs) in NeuraLUT terminology. These L-LUTs are then implemented directly in FPGA logic as a system of interconnected P-LUTs (Physical LUTs, such as 6-LUTs discussed in Section 2.2.1), enabling low-latency inference.

The key idea behind NeuraLUT is that each perceptron of the trained neural network is converted into its own lookup table that, in itself, represents the learned weights and biases as a trained, hidden two-layer MLP subnetwork. This results in a hardware structure in which the process of evaluating the prediction for a new unseen input, also known as inference, is reduced to a lookup in logic rather than arithmetic operations using different parameters. The tool flow of this conversion is illustrated in Figure 2.17.

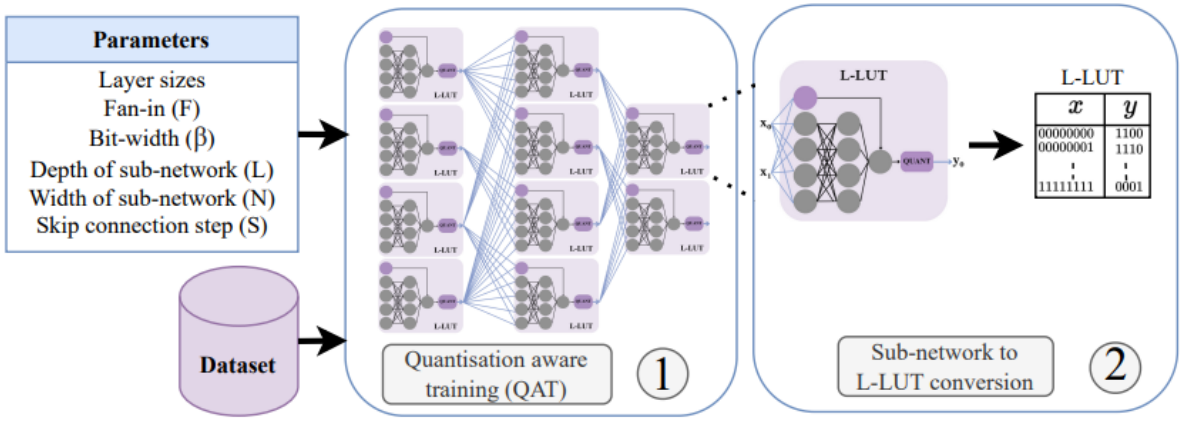


Figure 2.17: Visualisation of NeuraLUT's toolflow, adapted from Figure 4 in [17].

The resulting architecture is particularly suitable for FPGA implementation because configurable logic resources can implement lookup operations directly. During synthesis, even large tables are simply mapped across required logic blocks, with interconnecting paths between them, to ensure that all input combinations produce the required output. This direct mapping of the lookup tables avoids runtime arithmetic, reducing latency and potentially increasing power efficiency. However, converting neural

networks into LUTs, particularly for integration into FPGA logic blocks, introduces additional hardware constraints.

Specific Hardware Constraints

The main limitation of LUT-based neural network representations is the required input size of the lookup table. A Boolean LUT directly maps an input combination to an output value, meaning a LUT with n input bits requires 2^n stored outputs, where each output also consists of m bits. Consequently, increasing the input precision of the quantisation, in other words, requiring more bits to represent a decimal value, results in an exponential increase in memory requirements.

To address this issue, NeuraLUT uses small input widths, referred to as limited fan-in, in which each LUT accepts a limited number of input variables and, thus, input bits. Since in a multi-layer perceptron network each perceptron is at least connected with two preceding perceptrons, the choice for the resolution per input has to be carefully considered. Additionally, prior normalisation of the dataset should be applied to ensure that the full numeric range is efficiently mapped to the available binary representation. For example, an unsigned binary value 10010 would normally represent the number 18, but by applying a scaling factor during normalisation, it could instead represent a fractional value such as 0.576.

The internal network parameters and activations are handled during training and conversion in software, where the learned input-output behaviour is transformed into LUT contents using the selected input and output quantisation precision. Therefore, the hardware implementation does not require explicit storage or evaluation of the original fractional-valued network parameters during hardware inference.

Additionally, converting the neural network to LUT representations removes the original network's adaptability. Since the trained parameters are embedded into the generated LUT contents, changing the learned function requires retraining the model and regenerating the LUT representation before deployment. Therefore, the hardware implementation is optimised for efficient inference of a fixed function rather than on-device retraining or dynamic parameter updates.

2.3.3. LUT Compression

Although NeuraLUT reduced the computational cost of neural-network inference by requiring only lookup tables, the resulting LUT size remains dependent on the input precision. For higher-precision signals, the exponential growth of the truth table can make direct LUT implementation inefficient due to excessive logic block or even BRAM requirements. Therefore, additional compression techniques are needed to reduce hardware resources.

ReducedLUT

The ReducedLUT tool [18] introduces a method for compressing Boolean lookup tables by exploiting redundancy in the generated truth tables. Instead of storing every possible input-output combination, ReducedLUT identifies repeated patterns and represents them using compact representations. The tool flow of ReducedLUT is shown in Figure 2.18.

The core idea is to decompose the original large LUT into smaller, analyzable subtables. Identical subtables are represented only once and reused, reducing redundant storage at the cost of additional selection logic required to choose and reconstruct the corresponding output. Furthermore, subtables that are not identical but share a common structure can also be shared by storing only the information required to reconstruct the original outputs, such as a bit-shift as illustrated in Figure 2.19, offset, or lower-bit representation. By exploiting these repeated patterns, ReducedLUT avoids explicitly storing input-output combinations; instead, it stores the remaining unique subtables along with tables of shifts, biases, and index numbers that ensure the correct shift or bias is applied to reconstruct the original value. A visual algebraic analogy for this process is as follows. Where a conventional LUT stores an output y for every input, ReducedLUT instead searches for operators that are stored for their original value, and reconstructs it via the algorithmic expression:

$$y = ((u << t) + b, lb)$$

where u stands for the index of the original subtable, t represents the bit-shift needed, and b is the bias offset. The term lb denotes the stored low-bit segment, which is reattached to the shifted and biased value to reconstruct the final representation.

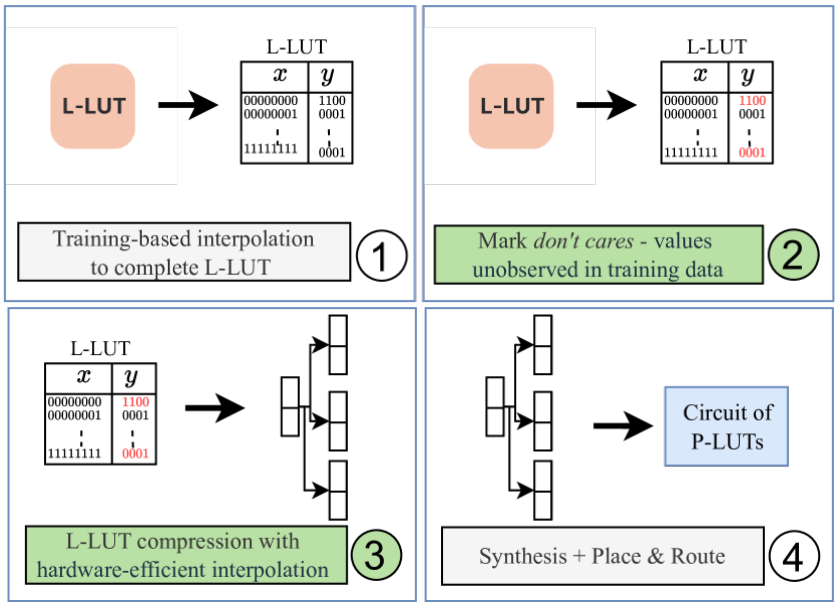


Figure 2.18: Visualisation of ReducedLUT's toolflow adapted from Figure 2 in [18].

Moreover, the last feature for redundancy exploitation, ReducedLUT, marks the input combinations that do not occur in the target application as *don't-care* values. Consequently, these entries act as wildcards, allowing the corresponding output values to be assigned in ways that enable more aggressive subtable matching while preserving the original functionality. An example of this process is shown by the red-highlighted values in Figure 2.19.

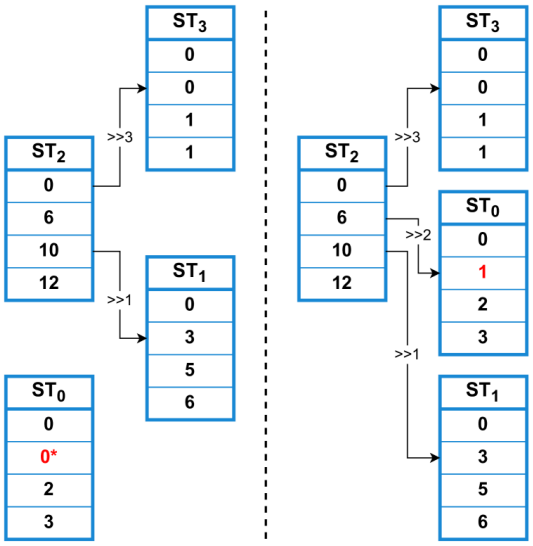


Figure 2.19: Visualisation of ReducedLUT's functionality of subtable matching and marking dontcare values as wildcards, adapted from Figure 1 in [18].

By exploiting unused input combinations and creative subtable matching, ReducedLUT can significantly reduce the hardware cost of LUT-based implementations. The effectiveness of these compression techniques depends on the contents and sizes of the subtables, as well as the number of values that can be marked as *don't care*.

2.4. Related Work

The implementation of Hodgkin-Huxley models on FPGAs has been extensively explored, targeting different trade-offs between biological fidelity, resource usage, and real-time performance. Most designs combine LUT-based logic, DSP blocks, and BRAM resources to evaluate the nonlinear differential equations. While these architectures achieve real-time operation across a range of neuron counts, the per-neuron hardware cost remains a key limiting factor for scalability, as nonlinear arithmetic operations require dedicated resources for each neuron update. This makes neuron density a critical constraint in resource- and power-limited applications, such as embedded or implantable neural devices.

At the high end of the design space, large-scale FPGA- and SoC-based systems, such as Virtex UltraScale+ implementations [45], and system-level platforms [46], demonstrate networks with millions of neurons. These results are achieved using substantially larger hardware resources and therefore reflect system-level scalability rather than efficiency under constrained FPGA budgets. In contrast, compact single-neuron implementations achieve highly efficient logic usage (e.g., $2k$ LUTs per neuron core [47]), but their scalability under large real-time network replication is not always explicitly evaluated. To enable a consistent comparison across this design space, Table 7.21 reports system-level characteristics, including FPGA resource usage, operating frequency, power consumption, and real-time neuron count. Table 2.4 complements this by reporting normalised per-neuron metrics, capturing the effective hardware cost of individual neuron implementations independent of scale or device capacity.

Work	Process Node (nm)	Device	#LUT	#DSP	Op.Freq. (MHz)	Power (W)	Model	Δt (μs)	Neurons @RT
ReLANCE [48]	28	Virtex 7 VC707	118,600	0	**198.4**	26.3	(CORDIC) HH	–	64
BiœmuS [49]	16	Zynq UltraScale+ KR260	29,300	468	–	6.5	**Pospischil** HH	31.3	**1,024**
Beaubois et al. [10]	16	Zynq UltraScale+ KR260	42,543	519	–	**5.6**	**Pospischil** HH	100.0	16
Zhang et al. [50]	90	Spartan-3A XC3SD1800A	**6,490**	**4**	100.0	–	HH	100.0	5
Yi et al. [51]	28	Zynq-7000	**8,457**	**45**	10.0	–	HH	**10.0**	11*
Akbarzadeh-Sherbaf et al. [12]	28	Artix-7 XC7A200T	**29,130**	280	71.4	–	HH	**7.8**	**4,096**
Khoyratee et al. [36]	28	Kintex-7 XC7K32T	167,346	840	100.0	–	HH	32.0	**15,000**
Bonabi et al. [52]	28	Virtex 7 XQ7K410T	86,032	1,112	63.4	**2.5**	HH	100.0	150
flexHH [53]	28	Stratix V 5SGSD8	–	–	**180.0**	40.1–46.8	HH	60.0	**10,368**
flexHH (full connectivity) [53]	28	Stratix V 5SGSD8	–	–	**180.0**	40.1–46.8	Extended HH	**10.0**	166
Zjajo et al. [54]	28	Virtex 7 XC7VX550	311,808	1,008	100.0	–	Extended HH	50.0	**1,188**
Smaragdos et al. [55]	40	Virtex 6 SX475T	190,464	**605**	150.0	–	Extended HH	50.0	330
Pourhaj et al. [56]	65	Virtex 5 V5-110t	43,388	–	**173.8**	–	Extended HH	10.0	**1,024**
Luo et al. [57]	28	Virtex 7 XC7VX485T	**32,142**	1,431	56.7	–	Extended HH	50.0	500

* Derived from reported latency: 100ms / 9.03ms execution time = 11.07 neurons at RT.

Table 2.3: FPGA implementations of Hodgkin–Huxley neuron models, showing resource usage (LUTs, DSPs, operating frequency, power) and model configuration (neuron model, timestep Δt, number of neurons, and FPGA device).

Work	#LUT/N	#DSP/N	Power/N (mW)	GFLOPS/N
Beaubois et al. [10]	2,658.9	32.4	350.0	–
ReLANCE [48]	1,853.1	0	411.0	12.69*
Zhang et al. [50]	1,298.0	0.80	–	–
Yi et al. [51]	763.7	4.06	–	–
Smaragdos et al. [55]	577.2	1.80	–	0.0748
Bonabi et al. [52]	573.5	7.40	16.7	–
Zjajo et al. [54]	262.5	0.84	–	2.13
Luo et al. [57]	64.3	2.86	–	763.7
Pourhaj et al. [56]	42.4	–	–	0.185
BiœmuS [49]	28.6	0.47	6.3	–
Khoyratee et al. [36]	11.2	0.056	–	–
Akbarzadeh-Sherbaf et al. [12]	7.2	0.068	–	–
flexHH [53]	–	–	3.9–4.5	11.22

* GOPS reported instead of FLOPS due to non-floating-point arithmetic.

Table 2.4: Comparison of FPGA Hodgkin–Huxley neuron implementations using normalised per-neuron metrics (LUTs, DSPs, power, GFLOPS), ordered by LUT usage per neuron

Several FPGA-based implementations demonstrate large-scale real-time Hodgkin-Huxley simulation. For example, flexHH [53] reports over 10,000 neurons on a Stratix-5 device, while Khoyratee et al. [36] achieve 15,000 neurons on a Kintex-7 FPGA. These results confirm that large-scale HH networks can be realised on FPGA hardware, but they depend on substantial device resources and therefore primarily reflect system-level scalability rather than neuron density under constrained resource budgets.

At the other end of the design space, compact implementations aim to minimise per-neuron cost. For instance, Zhang et al. [50] and Pourhaj et al. [56] optimise fixed-point arithmetic implementation enabling real-time single-neuron simulation with moderate replication potential. However, these designs still evaluate nonlinear HH dynamics using dedicated arithmetic units per neuron, which limits scalability when many neurons are instantiated under fixed FPGA resources. To reduce this cost, several works introduce approximation or alternative computation strategies. Akbarzadeh-Sherbaf et al. [12], for example, reduce LUT and DSP usage through approximation while maintaining real-time operation at scale, but still evaluate nonlinear functions at runtime.

Other implementations explore architectural alternatives for the underlying arithmetic. ReLANCE [48] replaces conventional DSP-based computation with a CORDIC-based design, reducing dependence on dedicated multipliers. SoC-based approaches such as BioEMU [49] and related work [10] integrate multicompartment neuron models into heterogeneous system architectures, while extended HH implementations [51, 54, 55, 57] explore different balances between model complexity, resource usage, and real-time performance across FPGA families.

Overall, existing work demonstrates real-time Hodgkin-Huxley simulation across a wide range of scales. However, a clear trade-off remains between scalability and per-neuron computational cost: large-scale systems require abundant resources, whereas compact implementations retain expensive nonlinear arithmetic, limiting replication within constrained FPGA budgets.

2.5. Conclusion

The FPGA implementations reviewed in this chapter cover compact single-neuron designs, intermediate architectures, and large-scale network platforms, forming a broad state-of-the-art landscape for real-time Hodgkin–Huxley simulation. Across these approaches, a consistent trade-off emerges between biological plausability, hardware resource usage, and achievable network size.

Large-scale systems achieve high neuron counts by leveraging high-end FPGAs or SoCs, but require substantial resources, limiting their use in resource-constrained settings. In contrast, compact implementations reduce per-neuron resource usage but still depend on explicit evaluation of nonlinear dynamics, which limits scalability through replication. Intermediate approaches partially bridge this gap using architectural optimisations and approximation techniques, but still rely on runtime evaluation of nonlinear functions via DSP-based or approximate arithmetic.

Overall, the state of the art indicates that although real-time Hodgkin–Huxley simulation is well established on FPGA platforms, achieving high neuron density under strict resource constraints remains a challenge. Current solutions either prioritise system-level scale or per-neuron efficiency, but rarely optimise both simultaneously.

This motivates alternative representations of Hodgkin–Huxley dynamics that reduce the per-neuron computational cost. Specifically, this thesis explores neural network-derived LUT representations as a means to improve achievable neuron density while maintaining biologically meaningful behaviour on FPGA hardware.

3

System Design

In this chapter, system architectures derived from the design challenges identified in Chapter 1 and Chapter 2 are investigated. The discussion begins with a discussion of the system requirements and evaluation metrics in Section 3.1.2, followed by the exploration of four architectural approaches with increasing levels of decomposition Section 3.2. For each approach, the underlying motivations and trade-offs are outlined, providing a basis for comparison. Based on these evaluations, selected architectures are further optimised towards efficient multi-neuron simulation.

3.1. Problem Analysis

The goal of this thesis is to develop an FPGA architecture capable of simulating biologically accurate Hodgkin-Huxley neurons while enabling scalability towards multi-neuron networks. As discussed in Section 2.1.1, the computational complexity of the HH model originates from its nonlinear dynamics, including coupled differential equations for the state-dependent evolution of each state at every simulation step. Conventional FPGA implementations typically address this challenge through arithmetic pipelines or dedicated function approximations. While these approaches enable real-time operation, their resource requirements increase significantly when scaling to larger neuronal networks, thereby limiting their use on resource-constrained FPGA devices.

LUT-based representations provide an alternative by replacing mathematical functions with precomputed mapping. However, directly converting the HH dynamics introduces a new trade-off, since the required storage grows exponentially with increasing input precision. Therefore, the central design challenge is to find an efficient decomposition of the computations that balances biological accuracy, hardware resource usage, and scalability.

In response to this challenge, this chapter discusses the concrete system requirements for the possible architectures and establishes the final benchmarking metrics by which this work is evaluated.

3.1.1. Resolution

As was described in Section 2.3.2, hardware implementations of Look-Up Tables and Neural Networks benefit from representing floating-point values as integers through normalisation and quantisation. During quantisation, the numerical range of a variable is divided into discrete intervals, where each interval is mapped to an integer representation using a fixed number of bits. Figure 3.1 illustrates the quantisation of a nonlinear function, where the finite number of representable values introduces a rugged approximation of the original function. A lower quantisation resolution increases the spacing between two represented numbers, resulting in a coarser, stair-case-like approximation as shown in the figure.

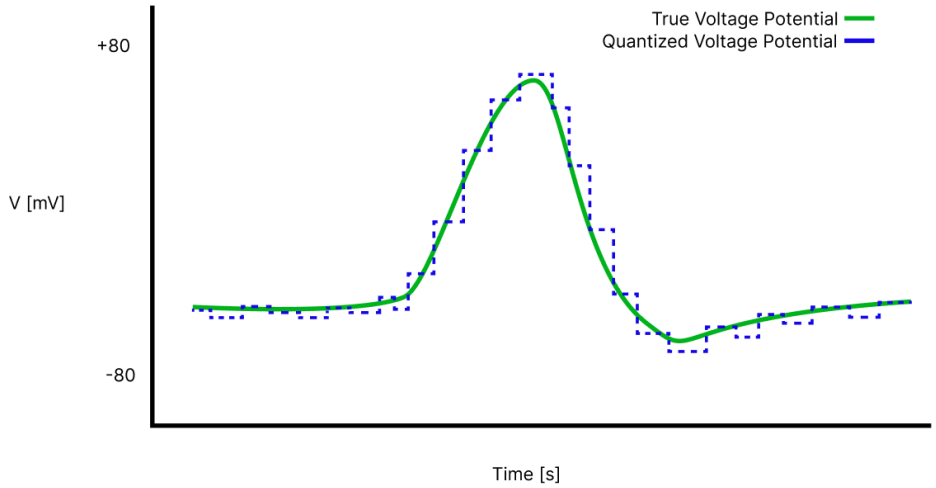


Figure 3.1: Simplified quantisation inaccuracy example showcasing how an insufficient resolution can lead to a staircase-like approximation

As described in Section 2.1.1, the HH-model exhibits rapid spike transitions with a slow recovery process. This implies that the discretisation of the state variables must provide sufficient resolution to capture these fast changes in voltage and gating variables. A 12-bit fixed-point format is adopted as the baseline representation, motivated by Lennart Landsmeer's (PhD with the NCL at Erasmus MC) *LUT_HH* software implementation [58], where this resolution was shown to preserve stable and accurate spike generation for a single HH neuron.

This 12-bit requirement, however, applies only to the discretised state variables (V, n, m, h) used during numerical integration. The nonlinear rate equations (α and β from (2.5)–(2.7)) contain exponential and division operations, including terms such as $\frac{x}{1e^x}$, which require substantially higher numerical precision due to the denominator approaching zero as $x \to 0$. Since the proposed LUT-based implementation stores only the mapping of the voltage and gate-channel equations, rather than evaluating every internal component online, the FPGA needs to store only the inputs and outputs of these quantised variables, for which a 12-bit fixed-point format is sufficient.

When extending to interconnected neurons, quantisation effects may accumulate through recurrent or feedback connections, potentially amplifying small numerical deviations. This network-level behaviour is not evaluated at this stage and is instead addressed in the system-level discussion of multi-neuron architectures in later chapters.

3.1.2. Functional Requirements

As introduced in Chapter 1, the primary challenge of the neuron simulation is to implement the biologically accurate but complex Hodgkin-Huxley model in a manner that remains scalable for large multi-neuron networks on resource-constrained FPGA platforms while minimising power consumption. To realise these design goals, they are translated into the high-level, system-wide objectives shown in Table 3.1.

Goal	Objective
Biological Accuracy	Long-term stability
	Real-time operation
Scalability	High neuron throughput
	Low resource utilisation
Low-power	Reduced computation complexity

Table 3.1: Design goals and corresponding design objectives

The high-level objectives related to resource utilisation and computational complexity can be translated into hardware design constraints, specific to our FPGA platform. Computational complexity is reduced by replacing costly arithmetic operations with resource-efficient alternatives, thereby minimising DSP utilisation. Low resource utilisation is divided into minimising the remaining FPGA blocks: Logic Block usage and BRAM (memory) usage. Consequently, the system-wide objectives are mapped to the system's two main architectural components: the neuron core and the neuron-core wrapper. The former performs the Hodgkin-Huxley state update. Therefore, it focuses on requirements related to the efficient hardware usage:

- Long-term stability
- Efficient logic utilisation
- Reduced DSP utilisation

The latter manages the storage and scheduling of multiple neurons, enabling the realisation of a neuronal network. Its requirements become as follows:

- Real-time operation
- High neuron throughput
- Efficient memory utilisation

Since the neuron core defines the computational cost of each neuron update, optimising this component is the primary focus before addressing network-level scheduling and resource sharing. To this end, four architectures with progressively increasing levels of decomposition are presented and evaluated with respect to the aforementioned requirements, which are now split and adopted into benchmarking metrics for the core and the wrapper, respectively:

Neuron Core

- Stability: The numerical stability of the neuron model, particularly when using Neural Network-based approximations, as approximation errors may vary between designs.
- Logic Usage: The hardware LUT utilisation resulting from different architectural decompositions and optimisation strategies.
- DSP usage: The utilisation of dedicated arithmetic resources, where reducing DSP dependency is a key objective compared to conventional HH implementations.

Neuron-Core Wrapper

- Real-time capability: the network must complete all neuron updates within the selected simulation timestep (0.01 ms), which represents a trade-off between numerical accuracy and simulation speed.
- Throughput: the number of neurons the network can handle within the real-time operation constraint
- BRAM usage: the number of BRAM blocks used to store all the states and connected synapses of all the neurons

In this work, stability refers to the preservation of the expected Hodgkin-Huxley dynamics after hardware transformation and approximation. This is particularly relevant for Neural Network-based LUT representations, where approximation errors can vary between designs and accumulate during iterative state updates. Stability is therefore assessed by verifying that the membrane voltage and gating variables remain bounded and retain biologically meaningful behaviour compared to the reference HH model.

Hereafter, in Chapter 4, optimisations for the four neuron core architectures are discussed and evaluated using the defined metrics. The neuron-core wrapper and its associated proof-of-concept network-level implementation are addressed separately after the final neuron core architecture has been established and validated.

3.2. Design Candidates

The hardware implementation of the HH neuron model constitutes a high-dimensional optimisation problem in which numerical stability, biological accuracy, FPGA resource utilisation, and scalability must be balanced simultaneously. To enable systematic design exploration, the main HH equation Equation 2.1 is decomposed into hardware-efficient computational components, each comprising derivatives, exponentials, and nonlinear functions. Rearranging the full equation for the capacitive term gives:

$$C_m \frac{dV_m}{dt} = I - \bar{g}_K n^4 (V_m - V_K) - \bar{g}_{Na} m^3 h (V_m - V_{Na}) - g_l (V_m - V_l) \tag{3.1}$$

Through the forward Euler method, explained in detail in Section 2.1.2, the next state calculation from the derivative is defined as:

$$V_{m,\text{next}} = V_m + \frac{dV_m}{dt}\Delta t, \tag{3.2}$$

$$n_{\text{next}} = n + \frac{dn}{dt}\Delta t, \tag{3.3}$$

$$m_{\text{next}} = m + \frac{dm}{dt}\Delta t, \tag{3.4}$$

$$h_{\text{next}} = h + \frac{dh}{dt}\Delta t, \tag{3.5}$$

where the α and β voltage-dependent rate functions are given by (2.5)–(2.7).
By substituting the voltage derivative from Equation (3.1) and the gate channel derivatives from (2.2)–(2.4) into (3.2)–(3.5), we get the full next state derivations for the Hodgkin-Huxley model:

$$V_{\text{next}} = V_m + \left[I - \bar{g}_K n^4 (V_m - V_K) - \bar{g}_{Na} m^3 h (V_m - V_{Na}) - g_l (V_m - V_l)\right] \cdot \Delta t / C_m \tag{3.6}$$

$$n_{\text{next}} = n + [\alpha_n(V_m)(1-n) - \beta_n(V_m) n] \cdot \Delta t \tag{3.7}$$

$$m_{\text{next}} = m + [\alpha_m(V_m)(1-m) - \beta_m(V_m) m] \cdot \Delta t \tag{3.8}$$

$$h_{\text{next}} = h + [\alpha_h(V_m)(1-h) - \beta_h(V_m) h] \cdot \Delta t \tag{3.9}$$

The exploration begins with the most straightforward and mathematically intuitive approach: replacing the entire HH model in Equation 3.6 with a single LUT. Although this represents the most direct LUT-based implementation of the model, the resulting LUT dimensionality grows prohibitively large, approaching the exabytes (10^{18}) range, and immediately revealing the fundamental scalability limitations of naive LUT substitution, as we will show in Section 3.2.1. This observation motivates progressively decomposing the mathematical model into smaller functional blocks, yielding increasingly fine-grained architectures. Simplified block diagrams of each design are presented in Figure 3.2. The following subchapters present the designs in order of increasing decomposition:

Architectures:

- **Design 1 - MONOLITH**: one monolithic LUT implementation containing the complete HH model;
- **Design 2 - SPLIT**: splitting of gating-variable dynamics from the membrane-voltage dynamics;
- **Design 3 - IONIC-SUM**: decomposition into separate ionic current blocks (I_K, I_{Na}, I_{leak}) which then are summed together;
- **Design 4 - HYBRID**: further decomposition of the ionic-current blocks into small LUTs combined with a limited number of multiplications.

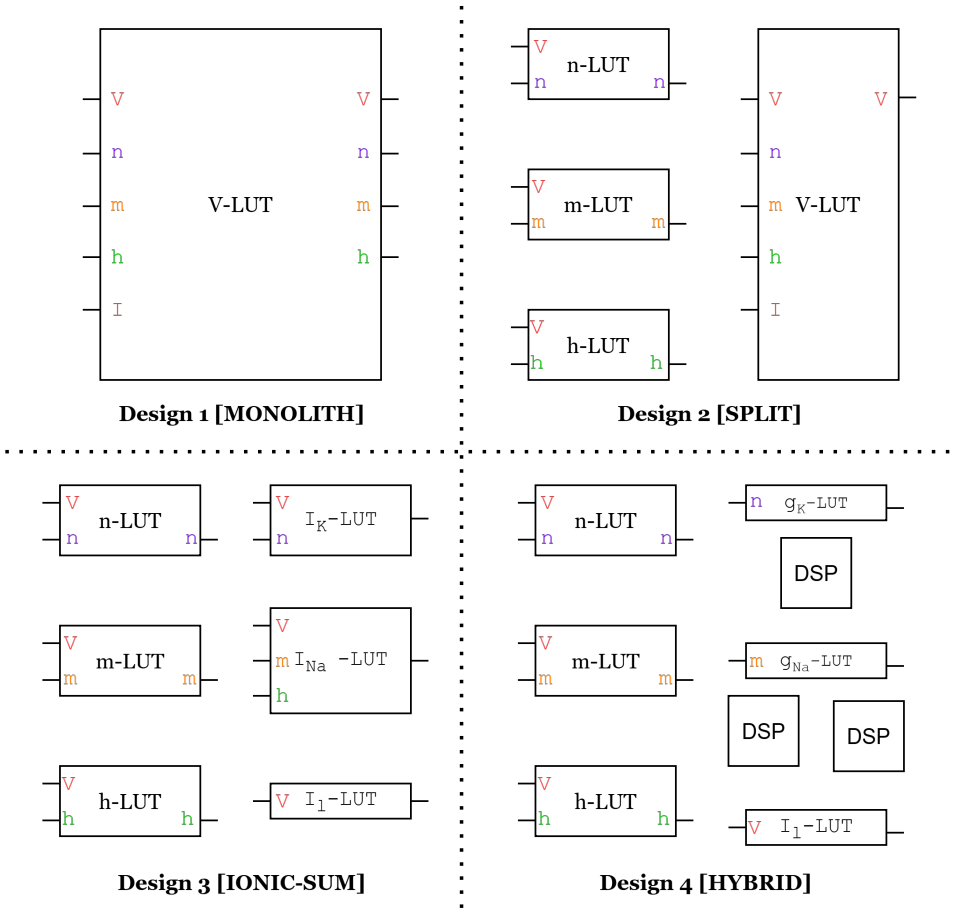


Figure 3.2: Overview of the four neuron-core architectures, illustrating the decomposition progression, from the coarse-grained 'MONOLITH' architecture to the progressively fine-grained 'SPLIT', 'IONIC-SUM', and 'HYBRID' architectures

3.2.1. Design 1 [MONOLITH]: 1 unified LUT

Introduction

The baseline for implementing the HH model in hardware explores the feasibility of replacing complex non-linear dynamics with Look-Up Tables to map all possible input combinations to their respective outputs.

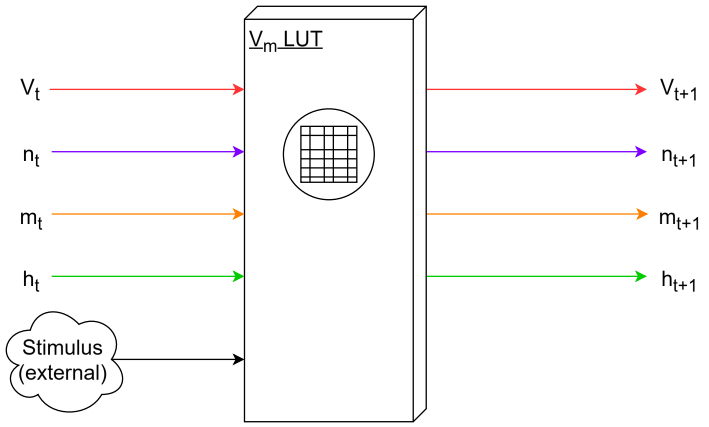


Figure 3.3: Design 1 [MONOLITH]: 1 unified LUT mapping all possible inputs of the HH model V, n, m, h, I to the outputs that are needed in the next step's update V, n, m, h

The initial architecture explores the most direct hardware realisation of the Hodgkin–Huxley (HH) model by replacing the full system of differential equations with a multidimensional lookup table (LUT).

Accordingly, the neuron update process can be reformulated as a direct state-transition mapping:

$$(V_m, n, m, h, I) \;\rightarrow\; \mathrm{LUT} \;\rightarrow\; (V_{\text{next}}, n_{\text{next}}, m_{\text{next}}, h_{\text{next}}) \tag{3.10}$$

In hardware, this approach removes the need for real-time evaluation of nonlinear differential equations, replacing it with a memory-access operation. The LUT therefore acts as a precomputed state-transition operator, enabling low-latency neuron updates at the expense of increased memory requirements. A small visualisation of such a LUT is shown in the figure below.

Design 1 LUT:
12 bits per variable → $2^{5 \cdot 12} = 1.2 \cdot 10^{18}$ entries

V	n	m	h	I	V_{next}	n_{next}	m_{next}	h_{next}
0...00	0...00	0...00	0...00	0...00	1...01	0...10	0...00	1...01
0...00	0...00	0...00	0...00	0...01	1...00	0...10	0...00	1...10
0...00	0...00	0...00	0...00	0...10	1...01	0...10	0...01	1...10
...					...			

Figure 3.4: Visual example of the LUT designed for Design 1 (MONOLITH)

However, the memory requirements scale exponentially with input dimensionality. The total storage S is calculated as:

$$S = 2^{(N_{in} \cdot w_{in})} \cdot (N_{out} \cdot w_{out}) \tag{3.11}$$

where N is the number of variables and w is the bit-precision. With 12-bit precision and five input variables, the LUT maps $2^{12 \cdot 5} = 2^{60}$ input combinations to four 12-bit output states, corresponding to approximately $S = 2^{60} \cdot 4 \cdot 12 \div 8[bits/byte] = 6.9$ exabytes of storage. Even if a trade-off would be done with the accuracy, and for example, a resolution reduction to 8 bits would still require approximately 4.4 terabytes.

Requirement Check

The expected characteristics of Design 1 with respect to the neuron-core metrics are visualised in Figure 3.5.

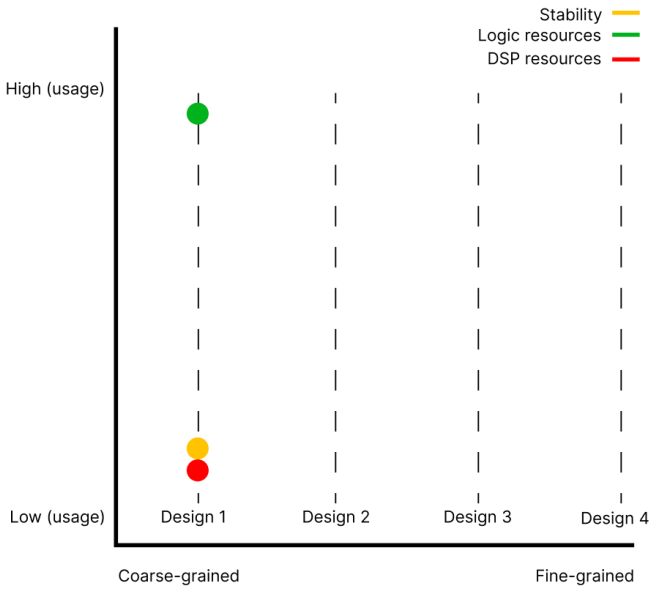


Figure 3.5: Design-space position of Design 1 (MONOLITH) across neuron core metrics

As expected, the fully LUT-based implementation achieves the objective of minimising DSP utilisation, since all dynamics are performed via input- and output-only lookup operations. However, the monolithic LUT structure results in poor logic-resource efficiency due to the exponential growth of dimensionality. Finally, as will be seen later in the results, such a large LUT negatively affects the stability of the resulting neuron dynamics.

Therefore, although the MONOLITH approach is mathematically the most direct LUT-based formulation and demonstrates the feasibility of eliminating DSP usage, its poor logic scalability and expected stability limitations make it unsuitable as a final architecture. This motivates further decomposition of the HH model before additional optimisation.

3.2.2. Design 2 [SPLIT]: Separate Gate LUTs and Voltage LUT

Introduction

Design 2 reduces the dimensionality of the monolithic LUT by separating the gating-variable dynamics from the membrane-voltage update. Instead of embedding all state transitions in a single mapping, the architecture is split into independent update functions for each gating variable and the voltage state. As opposed to equations (3.7)–(3.9) for a computational design, the gate-channel transitions using the proposed LUTs are now mapped as follows:

$$n_{\text{next}} = LUT_n(V_m, n), \quad m_{\text{next}} = LUT_m(V_m, m), \quad h_{\text{next}} = LUT_h(V_m, h)$$

Splitting the gate-channel variables from the main block means the voltage-membrane LUT no longer needs to determine those variable states, leaving the voltage state as the only LUT output. This separate function still depends on the full system state, however. The state transition from Equation 3.10 is reformulated as:

$$(V_m, n, m, h, I) \rightarrow \text{LUT}_\text{V} \rightarrow V_{\text{next}} \tag{3.12}$$

$$(V_m, n) \rightarrow \text{LUT}_\text{n} \rightarrow n_{\text{next}} \tag{3.13}$$

$$(V_m, m) \rightarrow \text{LUT}_\text{m} \rightarrow m_{\text{next}} \tag{3.14}$$

$$(V_m, h) \rightarrow \text{LUT}_\text{h} \rightarrow h_{\text{next}} \tag{3.15}$$

This decomposition introduces some unequal workload balance. The gating LUTs become independent low-dimensional lookup structures, while the voltage LUT remains complex and expects the full combination of all state variables (V_m, n, m, h, I) as input, thereby remaining excessively large. Numerical estimation shows:

$$S = 2^{(5\cdot 12)} \cdot (1 \cdot 12) \div 8 \approx 1.7EB$$

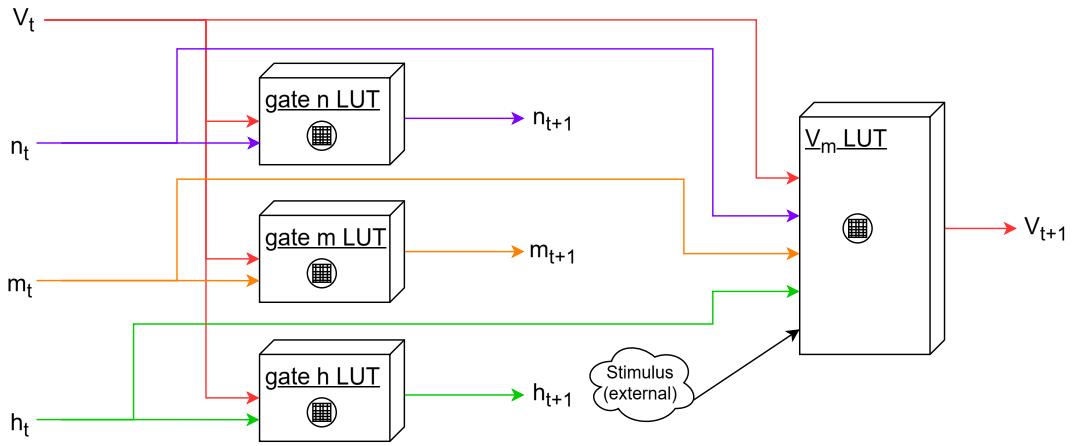


Figure 3.6: Design 2 [SPLIT]: separation of gate-channel LUTs and voltage-membrane LUT

One gate-channel LUT, without any optimisation, would theoretically only cost around 25 MB of FPGA memory, which is at least something computers can create and test, in contrast to the large unified voltage LUT from the previous design:

$$S = 2^{2 \cdot 12} \cdot 1 \cdot 12 \div 8 \approx 25.2MB$$

However, the corresponding RTL files in Verilog, generated by mapping all input combinations to their outputs, occupy up to 568 MB. Even if one were to map only the outputs in hex format, a single-gate hex file would cost 64MB ($2^{24} \cdot 4$ bytes (3 digits + newline per entry)), which is still heavy for a simulator such as Vivado to process.

Requirement Check

The expected characteristics of Design 2 with respect to the neuron-core metrics are visualised in Figure 3.7.

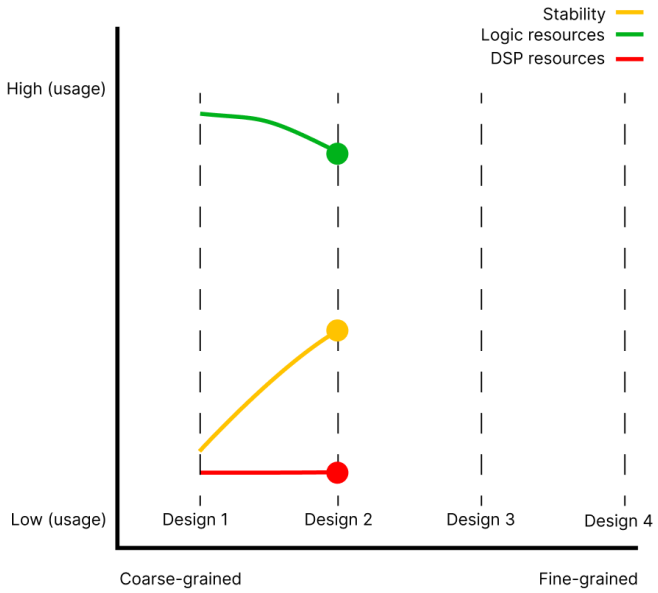


Figure 3.7: Design-space position of Design 2 (IONIC-SUM) across neuron core metrics

Compared to design 1, logic resource utilisation is reduced primarily due to the output side of the voltage LUT. While three additional LUTs are introduced for the gate channels, their contribution is relatively small compared to the reduction achieved in the voltage LUT. The three gate LUTs together require $(3 \cdot 2^{24} \cdot 12$ bits$)$, corresponding to roughly 600Mbits of storage. In contrast, the voltage LUT output width is reduced from four state variables to a single state variable, reducing its storage requirement from $2^{60} \cdot 12$ bits $\approx 55 \cdot 10^{18}$ bits to $2^{60} \cdot 12$ bits $\approx 14 \cdot 10^{18}$ bits. This corresponds to a reduction of approximately 41 exabits, making the additional gate-channel LUT cost negligible by comparison.

DSP utilisation remains unchanged, as the arithmetic operations required for both the membrane-voltage and gate-channel computations are not modified at this stage of the design exploration.

Numerical stability shows a modest improvement compared to the first design. By separating the gate-channel updates into independent mappings, each LUT can be optimised for the dynamics of a single state variable rather than approximating the behaviour of the complete system simultaneously. This allows the individual subsystems to more closely match their target functions, leading to improved overall behaviour.

Despite these improvements, the voltage LUT remains the dominant constraint in the design, as it still depends on the full system consisting of 5 input variables. This limitation motivates further decomposition of the voltage computation in the following designs.

3.2.3. Design 3 [IONIC-SUM]: gate LUTs and summed-current LUTs

Introduction

This decomposition significantly reduces LUT dimensionality by replacing a single five-dimensional voltage mapping with several lower-dimensional ionic-current mappings. Unlike design 2, where the voltage LUT still depended on the complete state vector (V_m, n, m, h), each conductance term is now determined independently according to its own physical dependencies.

Comparing with the original next-state calculation for the membrane voltage, Equation 3.6 is now decomposed into ionic current subparts, transforming the next-state calculations to:

$$\begin{aligned} V_{\text{next}} &= V_m + \left[I - \bar{g}_K n^4 (V_m - V_K) - \bar{g}_{Na} m^3 h (V_m - V_{Na}) - g_l (V_m - V_l)\right] \cdot \Delta t / C_m \\ &= V_m + [I - I_K - I_{Na} - I_{\text{leak}}] \cdot \Delta t / C_m \\ &= V_m + I \cdot \Delta t / C_m + dV_K + dV_{Na} + dV_{\text{leak}} \end{aligned} \tag{3.16}$$

where

$$dV_K = -\left[\bar{g}_K n^4 (V_m - V_K)\right] \cdot \Delta t / C_m, \tag{3.17}$$

$$dV_{Na} = -\left[\bar{g}_{Na} m^3 h (V_m - V_{Na})\right] \cdot \Delta t / C_m, \tag{3.18}$$

$$dV_{\text{leak}} = -\left[g_l (V_m - V_l)\right] \cdot \Delta t / C_m. \tag{3.19}$$

Instead of directly predicting the next voltage state, the conductance currents are derived independently and then subtracted from the stimulus injection I. Since the LUTs are meant as a replacement for expensive arithmetic operations, each ionic combination $I_i \cdot dt/C_m$ is replaced by a LUT, instead of only I_i. The state-transition from Equation (3.12) becomes:

$$(V_m, n) \;\rightarrow\; \mathrm{LUT_K} \;\rightarrow\; dV_K \tag{3.20}$$

$$(V_m, m, h) \;\rightarrow\; \mathrm{LUT_{Na}} \;\rightarrow\; dV_{Na} \tag{3.21}$$

$$V_m \;\rightarrow\; \mathrm{LUT_{leak}} \;\rightarrow\; dV_{leak} \tag{3.22}$$

This decomposition significantly reduces LUT complexity because each LUT now models a smaller physical process. Additionally, the architecture becomes modular and extensible for future ion-channel additions.

The largest remaining LUT in the architecture becomes the sodium-current LUT (I_{Na}), which depends on three state variables (V_m, m, h), while the potassium (I_K) and leak (I_{leak}) contributions depend only on two and one input variables, respectively. Using the storage estimate from Equation 3.11, the sodium LUT requires

$$S = 2^{3 \cdot 12} \cdot 12 \div 8 \approx 103 \text{ GB}.$$

This represents a substantial reduction compared to the exabyte-scale voltage LUT required in Design 2. Theoretically, this could already be doable in high-end hardware. However, as mentioned earlier, the RTL files always end up much larger, making them impractical for implementation within the available toolchain.

The resulting design architecture for this decomposition is visualised in the block diagram in Figure 3.8.

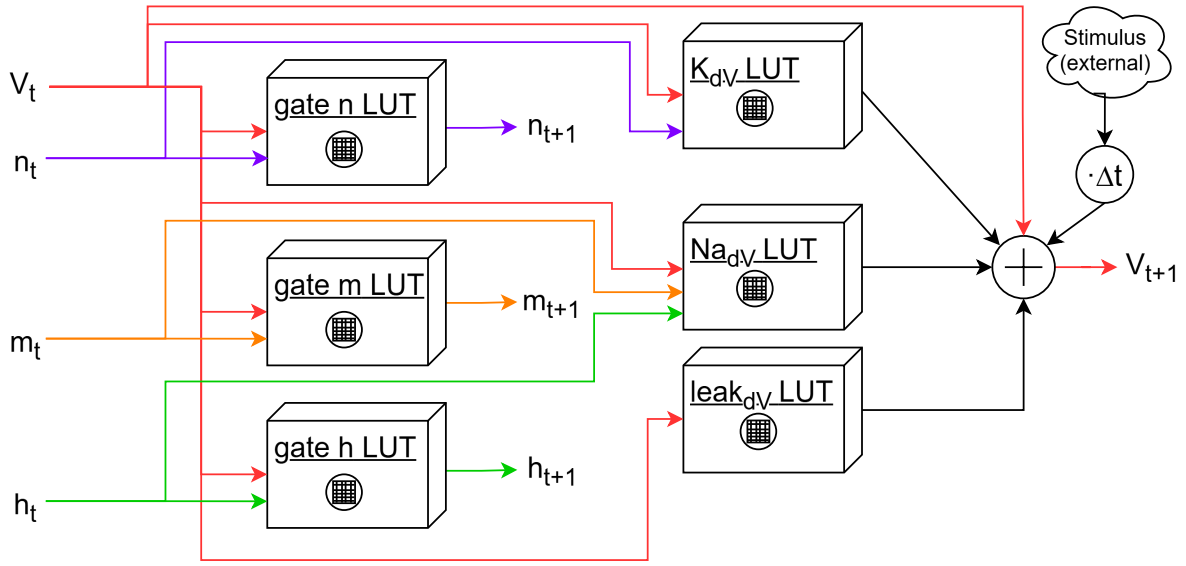


Figure 3.8: Design 3 [IONIC-SUM]: separate LUTs for gate-channel dynamics and ionic-current contributions, which are summed to determine the membrane-voltage update

Requirement Check

The expected characteristics of Design 3 with respect to the neuron-core metrics are visualised in Figure 3.9.

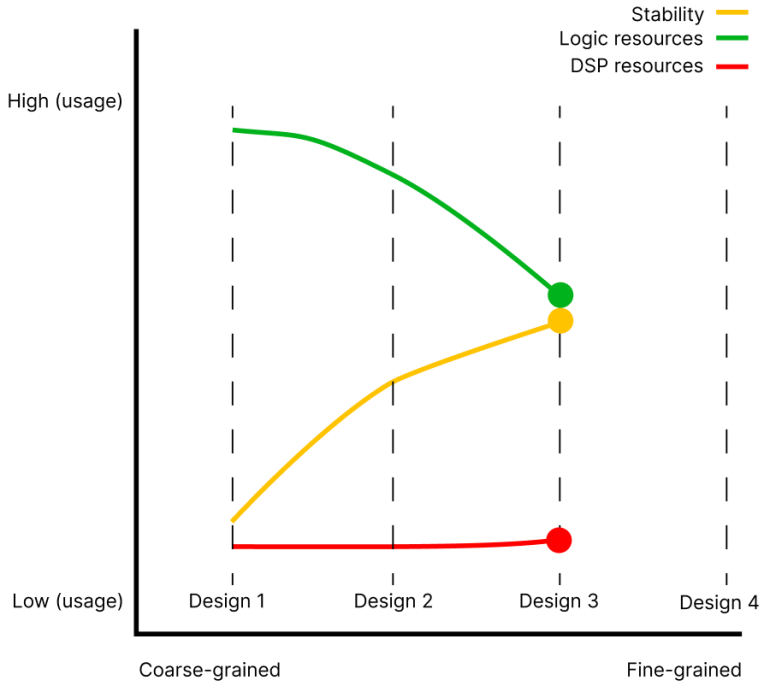


Figure 3.9: Design-space position of Design 3 (IONIC-SUM) across neuron core metrics

Compared to Design 2, logic resource utilisation decreases significantly again, due to the decomposition of the voltage LUT. The dominant voltage LUT from Design 2 required 1.7 exabytes, formally written as $1.7 \cdot 10^{18}$ bytes, whereas the largest LUT from the current design uses only 103 gigabytes, formally written as $1 \cdot 10^{11}$ bytes. This represents a reduction of 7 orders of magnitude, the largest improvement in logic usage achieved thus far in the design exploration.

Since all multiplications are still absorbed by LUT mappings, and additions do not need any DSP blocks, no change in DSP utilisation is to be observed for this design.

Numerical stability improves further because each LUT is responsible for determining a smaller, more biophysically realistic part of the Hodgkin-Huxley model. Reduced functional complexity reduces the approximation error of larger LUTs, thereby allowing the underlying dynamics to be represented more accurately.

In the end, Design 3 demonstrates that decomposing the membrane-voltage equation into its related subparts is an effective strategy for reducing LUT complexity. However, before continuing with implementing optimisations on the designs, the remaining three-dimensional sodium-current (I_{Na}) LUT continues to dominate the architecture and therefore motivates further decomposition.

3.2.4. Design 4 [HYBRID]: separate gate and conductance LUTs + some multiplications

Introduction

Design 4 shifts the focus of decomposition towards selective arithmetic reconstruction. While the previous designs aimed to replace nonlinear arithmetic entirely with LUT-based representations, Design 4 deliberately relaxes this constraint. The motivation is that a small number of DSP-based multiplications may significantly reduce the required LUT storage, improving the overall resource efficiency of the architecture. Therefore, DSP usage is reintroduced as a design trade-off, in which limited arithmetic resources are traded for a substantial reduction in LUT complexity.

While Design 3 reduced the dimensionality of membrane-voltage computation by separating the ionic-current contributions, the sodium-current LUT remained excessively large because it required 3 input state variables. Rather than storing complete current mappings, Design 4 implements only the highly non-linear elementary terms, such as n^4 and constant variable multiplications, in LUT form. As a result, only LUTs with 1 input and 1 output variable remain for the current updates, while the remaining calculations use explicit arithmetic operations with DSP blocks. The partial voltage contributions from Equation 3.16 are translated into the following calculations:

$$\text{Potassium: } dV_K = -\text{LUT}_K(n) \cdot (V_m - V_K) \cdot \frac{\Delta t}{C_m}, \qquad \text{LUT}_K(n) = \bar{g}_K n^4 \tag{3.23}$$

$$\text{Sodium: } dV_{Na} = -\text{LUT}_{Na}(m) \cdot h \cdot (V_m - V_{Na}) \cdot \frac{\Delta t}{C_m}, \quad \text{LUT}_{Na}(m) = \bar{g}_{Na} m^3 \tag{3.24}$$

$$\text{Leak: } dV_{leak} = \text{LUT}_{leak}(V_m), \qquad \text{LUT}_{leak}(V_m) = -g_l(V_m - V_l) \cdot \frac{\Delta t}{C_m} \tag{3.25}$$

For the potassium and sodium channels, the terms $(\bar{g}_K \cdot n^4)$ and $(\bar{g}_{Na} \cdot m^3)$ are implemented as simple one-dimensional LUTs, since $\bar{g}_K$ and $\bar{g}_{Na}$ are constants and don't need to be applied as input variables. The remaining voltage-dependent and gating-dependent multiplications are performed using DSP resources.

This approach selects LUTs where they provide the greatest benefit, namely, for approximating highly non-linear functions, while allowing simpler arithmetic relationships to be computed in the regular way with DSPs, removing some sources of approximation errors. In contrast to the potassium and sodium, the leak-current contribution has remained as a direct LUT mapping. Since it depends only on the membrane voltage, the resulting LUT already contains a single input dimension and therefore gains little from further decomposition. Replacing it with additional arithmetic would increase implementation complexity without providing a meaningful reduction in memory requirements.

As a result, all remaining LUTs become either two-dimensional gate LUTs or one-dimensional conductance LUT mappings combined with arithmetic operations. This makes the architecture closer to

a synthesizable solution within the FPGA resource constraints, but without any optimisations that address LUT size, synthesis tests cannot yet be performed. Additionally, in further explorations of this design, extra care must be taken with the timing architecture, since DSP blocks introduce more delay than routed LUT paths. The resulting block diagram of this decomposed architecture is shown in Figure 3.10.

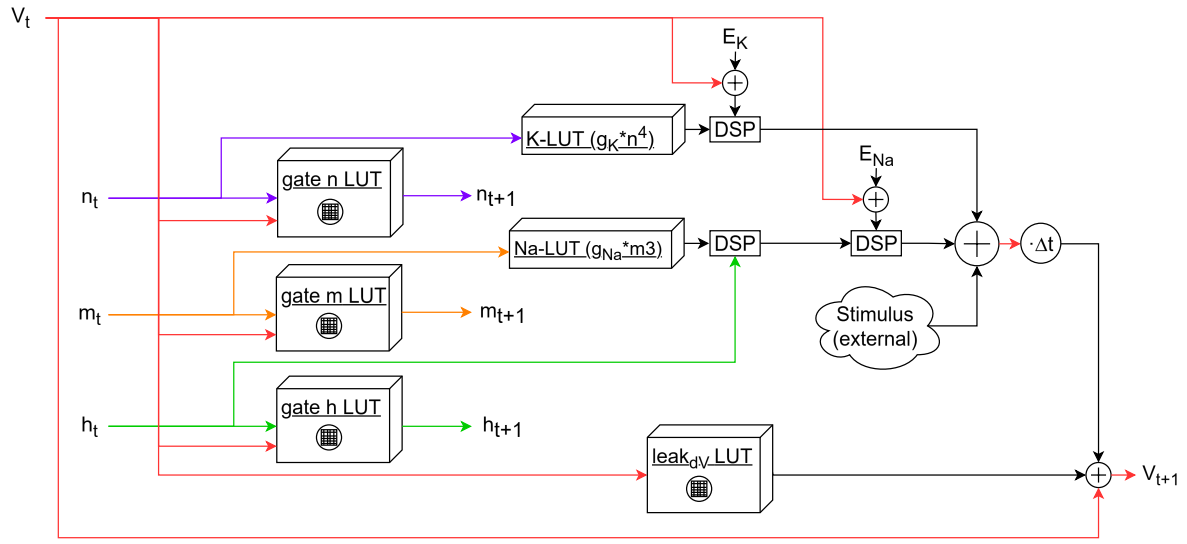


Figure 3.10: Design 4 [HYBRID]: two-dimensional LUTs for gate channel dynamics, one-dimensional conductance LUTs for $\bar{g}_K n^4$ and $\bar{g}_{Na} m^3$ combined with some arithmetic operations, and a leakage current I_{leak} LUT with already embedded timescaling

Requirement Check

The expected characteristics of Design 4 with respect to the neuron-core metrics are visualised in Figure 3.11.

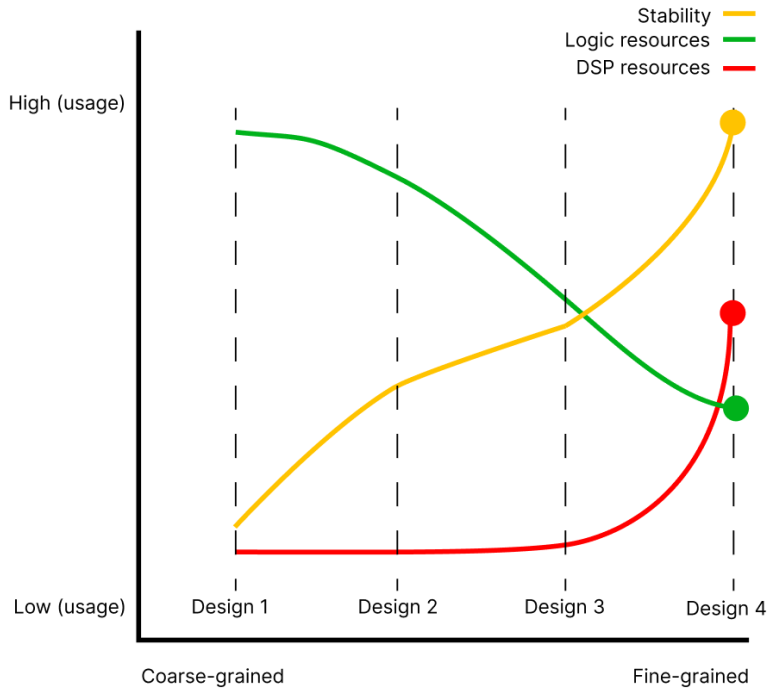


Figure 3.11: Design-space position of Design 4 (HYBRID) across neuron core metrics

Compared to Design 3, logic utilisation is substantially reduced because the remaining multi-dimensional current LUTs are replaced by a hybrid of simple conductance LUTs and arithmetical operations.

This reduction in LUT complexity, however, comes at the cost of increased DSP utilisation. Design 4 explicitly implements several multiplication stages using dedicated DSP resources. The increase remains modest relative to the available DSP budget of modern FPGAs, but in the metric figure, the penalty is set and visualised as being relatively high, as the objective was to have no DSP utilisation.

Numerical stability improves further because the dominant ionic-current contributions are no longer determined by large LUTs, but instead, the underlying mathematical structure is maintained through explicit arithmetic operations, reducing approximation error while maintaining the compact representation of the highly non-linear conductance terms.

3.3. Design Evaluation

Overall, Design 4 represents the first practical approach in this work to achieve a synthesizable balance between LUT complexity and arithmetic resource usage. A summarised overview of the designs, including their trade-offs, is shown in Table 3.2 and again visualised in Figure 3.12.

Design	Main idea
Design 1 - MONOLITH	Entire HH update represented by one unified LUT
Design 2 - SPLIT	Separate LUTs for gate variables and membrane voltage
Design 3 - IONIC-SUM	Gate LUTs combined with LUTs for summed ionic currents
Design 4 - HYBRID	Gate and conductance LUTs combined with selected arithmetic operations

(a) Architectural concepts

Design	Granularity	Arithmetic	Memory	Modularity	Scalability
Design 1 - MONOLITH	Very coarse	⋆⋆⋆⋆⋆	★★★★★	★⋆⋆⋆⋆	★⋆⋆⋆⋆
Design 2 - SPLIT	Coarse	⋆⋆⋆⋆⋆	★★★★⋆	★★⋆⋆⋆	★★⋆⋆⋆
Design 3 - IONIC-SUM	Fine	⋆⋆⋆⋆⋆	★★⋆⋆⋆	★★★★⋆	★★★★⋆
Design 4 - HYBRID	Very Fine	★⋆⋆⋆⋆	★⋆⋆⋆⋆	★★★★★	★★★★★

(b) Quantitative comparison

Table 3.2: Design space exploration of FPGA implementations of the Hodgkin–Huxley model, where the designs are additionally visualised in Figure 3.12

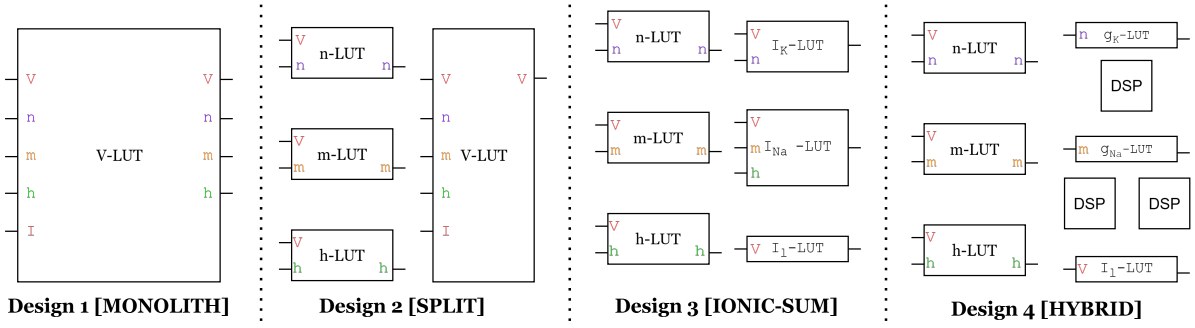


Figure 3.12: Simple overview of the four neuron-core architectures supplementary to Table 3.2

Although the reintroduction of limited DSP resources improves hardware feasibility compared to earlier fully LUT-driven approaches, LUT growth remains a limiting factor for scalable multi-neuron deployment. As a result, architectural decomposition alone is insufficient to fully resolve the resource bottleneck; ensuring further optimisation of how nonlinear terms are represented is required. This motivates the introduction of additional compression strategies in the next stage of the design space exploration, starting with the HYBRID architecture (Design 4) and extending to alternative decompositions where beneficial.

4

System Implementation

4.1. Overview

The implementation trajectory explored in this chapter follows a cumulative optimisation strategy, meaning a sequential progression of optimisation steps added to the neuron core architecture. Each optimisation step addresses a bottleneck identified in a preceding implementation while preserving all previously introduced optimisations. This creates a chronological optimisation path in which the impact of each modification can be evaluated incrementally.

Optimisations:

Step	Optimisation	Primary Goal	Key insight
1	LUT Compression	Remove redundant LUT entries using an adaptation of the ReducedLUT algorithm [18]	Major reduction logic/memory usage
2	Delta Prediction	Encode differences rather than complete next states, followed by Euler propagation	Improves accuracy and stability
3	NeuraLUT	Replace explicit LUTs with Neural Network-derived LUTs via the adapted NeuraLUT tool [17]	Major potential for reduction in logic/memory
4	Bit-Resolution Reduction	Reduce numerical precision in the input and hidden layers of the Neural Network-derived LUTs	Additional reduction in logic resources

Similar to the design exploration trajectory in the previous chapter, each optimisation is evaluated against the neuron core metrics. While multiple architectural candidates were explored during the design phase (Design 1-4), the optimisation process is not performed uniformly across all designs. Instead, Design 4 serves as the primary reference implementation for all optimisation experiments. This choice is motivated by its improved balance between LUT complexity and arithmetic feasibility, as well as its greatest suitability for synthesis with the available tools compared to the other designs. Moreover, the gate-channel LUTs associated with Design 4 are the same as those used in Designs 2 and 3, so Design 4 is a good indicator of whether an optimisation is sufficiently effective to apply to the other designs as well. Even though Design 4 serves as the evaluation baseline for the optimisation metrics, Designs 1 to 3 are also tested, where simulation and synthesis feasibility allow, to provide a complete overview. These cases are also used to verify whether observed effects are architecture-dependent or generalisable across different decomposition levels.

The evaluation metrics presented in each section are interpreted as relative comparisons across optimisation levels under a fixed reference architecture, rather than cross-architecture performance comparisons. Since Design 4 is the primary reference architecture, it determines the main trend shown in

the metric figures, while results from Designs 1 to 3 are included for evaluation only where feasible and where they provide additional value. Consequently, the visual differences in metric behaviour reflect the effects of successive, cumulative optimisations on a consistent reference design, rather than direct comparisons between different architectural decomposition levels. The evaluation of the baseline against these metrics is shown in Figure 4.1.

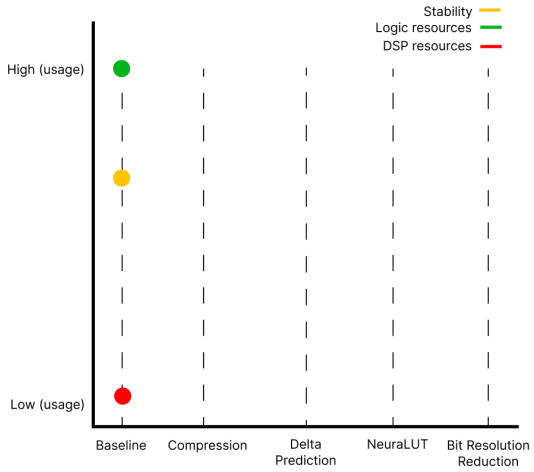


Figure 4.1: Evaluation of the baseline across neuron core metrics

The baseline implementation exhibits higher logic utilisation than the future optimisation levels, reflecting the absence of structural reduction or compression techniques in the optimisation chain. DSP utilisation remains low because no additional arithmetic structures are introduced beyond those defined in the underlying neuron core formulation. In terms of stability, the baseline provides the reference behaviour against which all optimised implementations are compared in the following sections, starting at a moderately high level because its only dependence is on LUT quantisation inaccuracies.

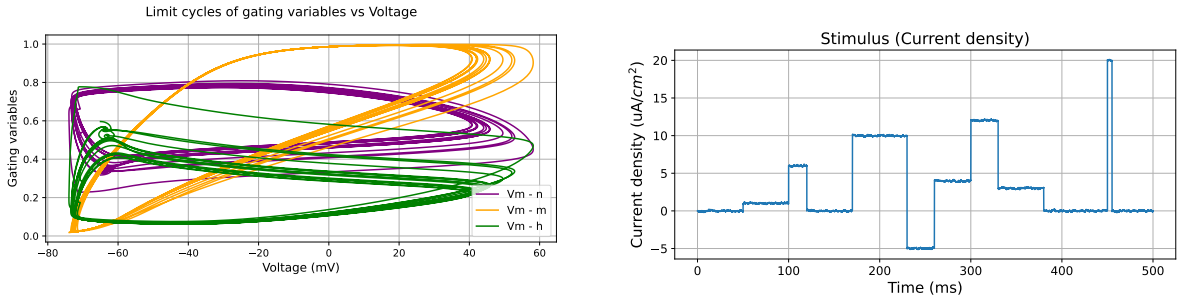
4.2. First Optimisation: LUT compression

A key limitation observed in the earlier design exploration is that direct implementation of the more coarse-grained, larger lookup-table-based architectures is not feasible within the available simulation and synthesis toolchain. In particular, generating full LUT representations is practical only for sufficiently low-dimensional mappings.

For this reason, the first optimisation applies the LUT compression library introduced in Section 2.3.3 to the most synthesis-feasible architecture from Chapter 3, namely Design 4 (HYBRID). Since this design contains the lowest-dimensional LUT structures among the explored architectures, it is the most suitable candidate for evaluating the effectiveness of compression. In contrast, Designs 1 and 2 (MONOLITH and SPLIT) are not even considered for this optimisation stage, as it was shown in Section 3.2.3 that the membrane-voltage LUTs scale up to 820Gbits for Design 2 and to the exabyte range for Design 1, which exceeds the practical storage limits within the used workflow.

4.2.1. Optimisation 1: Design 4 [HYBRID]

In Figure 4.2a, the ratio between the voltage and the gating variables is shown across a dataset of various stimuli, as defined in Figure 4.2b on the right. As can be seen, for each gating variable, there is a substantial area of unused space within and outside of the ellipses, representing the limit cycles of the voltage-gate channel ratio. Regardless of stimulus frequency or magnitude, the limit cycles stay roughly on the same trajectory. Recent work on LUT compression via a tool called ReducedLUT [18] demonstrated that introducing don't-care conditions, in other words, disregarding the unused input combinations, during LUT decomposition can significantly reduce FPGA LUT utilisation while maintaining negligible accuracy degradation, achieving up to a $1.63\times$ reduction in physical LUT usage [18].



(a) Simulated ratio between gating variables n,m,h and the voltage across the phase space, known also as limit cycles

(b) Used stimulus injection for the simulation in pane (a)

Figure 4.2: Limit cycles V-n, V-m, V-h across 500 ms of various injected stimuli

Based on these findings, ReducedLUT compression was investigated as a potential optimisation strategy for the LUTs in this work. The methodology, therefore, includes evaluation of whether such decomposition approaches can sufficiently reduce the memory footprint and synthesis cost of the LUT-based architecture to improve FPGA feasibility. The implementation includes a significant rewrite of the algorithm because it relies on thorough yet time-consuming nested loops to search for similar subtables to merge. Thus, the compression ratios, accuracy impact, and synthesis results are based on this proof-of-concept deviation from the original ReducedLUT, which now operates on 12-bit tables, unlike the original publication. The algorithm is explained in more detail in Chapter 5, after which the results of LUT compression for the Design 4 [HYBRID] LUTs are presented in Chapter 7.

Although prior work demonstrates substantial reductions in LUT utilisation, preliminary architectural analysis suggests that resource consumption may remain significant for large-scale Hodgkin-Huxley implementations due to the exponential growth inherent in LUT-based representations. Consequently, apart from hardware synthesis evaluation to determine whether compressed gate-channel LUTs can be accommodated within the resource constraints of standard FPGA devices, additional optimisations need to be investigated.

4.2.2. Optimisation 1: Design 3 [IONIC-SUM]

Design 3 was not explored extensively because its feasibility is largely influenced by the gate channels, which have already been evaluated. Nevertheless, testing the feasibility and accuracy of the combined behaviour of the gate-channel LUTs together with the (compressed) conductance-current LUTs could provide valuable insights for further decisions regarding the IONIC-SUM design.

Section 3.2.3 already showed that this design does not provide a feasible decomposition approach for the Sodium potential (I_{Na}) LUT, which requires a 3-variable input dimensionality. Since this is not feasible with the tools available for a 12-bit resolution per input variable, the bit width has to be reduced to 8 bits, so that the total input sum is 24 bits, thereby keeping the decomposition process computationally feasible on standard computing hardware. However, reducing the Sodium current LUT resolution to 8 bits per variable increases the quantisation error of the current approximation, as discussed in Section 3.1.1. This could lead to consequences, such as deviations in spike timing, reduced spike amplitude, or, in extreme cases, failure to reproduce expected spike behaviour. Therefore, any conclusions drawn from this design's results should account for the reduced resolution as a potential source of numerical error. A full-resolution Sodium LUT could theoretically be evaluated on hardware, but this would require substantially higher resource utilisation and is beyond the scope of this work.

Additionally, the compressed conductance LUTs still directly approximate the complete next-state behaviour, after which they are multiplied by Δt, as shown in the equations elaborated in Section 3.2.3. Since the conductance dynamics arise from coupled differential equations, it is relevant to investigate whether derivative-based prediction can yield improved numerical behaviour and recursive stability for the decomposed current mappings. Therefore, the next optimisation stage focuses on the amount of change (deltas) by storing derivatives rather than storing full next states in the LUTs.

4.2.3. Optimisation 1: Requirement Check

This optimisation focuses entirely on reducing logic usage. In theory, nothing will change for the stability levels, since an extensive margin is chosen for all LUT compressions, so that even input combinations that do not exist in the dataset used for the algorithm remain covered after compression. Lastly, nothing is implemented to affect the DSP resource usage.

4.3. Second Optimisation: Delta Prediction

The first optimisation reduced LUT complexity using a compression algorithm, but the resulting architectures still rely on direct next-state prediction, as it was considered the most straightforward solution. To illustrate quickly, the LUTs approximate the complete state transition as in this example for the n-gate:

$$(V_m, n) \rightarrow n_{t+1}.$$

Although this formulation is straightforward, it does not directly reflect the mathematical structure of the Hodgkin-Huxley model, which is defined by differential equations [59]. The evolution of the state variables is determined by their derivatives, and the next state is then obtained via integration or approximation, such as Euler propagation. For this reason, the second optimisation reformulates the LUT outputs from direct next-state predictions to derivative-based predictions. The state transition of the LUT becomes

$$(V_m, n) \rightarrow \frac{dn}{dt},$$

after which the next state is reconstructed through Euler propagation as explained in Section 2.1.2:

$$x_{t+1} = x_t + \frac{dx}{dt}\Delta t$$

The reformulation is motivated by a quantisation-precision argument. With the next-state prediction, each quantisation step of 12-bit resolution maps directly to a gate-state change, whereas with the derivative prediction, each step maps to a difference in the rate of change. Since the derivative represents the change per unit time, it must be multiplied by the simulation time-step Δt to obtain the actual state increment used in the Euler forward update. This allows the simulation to represent smaller incremental changes in the gate variables over time, leading to smoother state evolution without increasing the LUT quantisation width.

However, since the LUT interface remains quantised at 12-bit resolution, small-scale updates could be lost if directly re-quantised after each iteration. The reformulation is therefore also motivated by an accumulation argument. In the next-state formulation, each LUT output is directly quantised into a 12-bit state update, meaning that any change smaller than one least significant bit (LSB) is immediately lost due to truncation. In contrast, the state update is not directly written back into a truncated 12-bit representation after each iteration. Instead, updates are accumulated in a higher-precision (24-bit) register, which serves as the persistent state storage for subsequent time steps. The 12-bit LUT interface is used only to evaluate the derivative function, while the state evolution itself is performed in the higher-precision domain. This prevents repeated loss of sub-LSB contributions across iterations.

As a result, the system does not increase the intrinsic resolution of the LUT itself, but improves the effective dynamic range of the state evolution for small derivative contributions; therefore, this optimisation is expected to exhibit smoother local behaviour and stronger correspondence to the original Hodgkin-Huxley equations.

To observe the effect of the reformulation, the optimisation is first applied on the HYBRID architecture, Design 4, because it contains the simplest LUT structures and was previously identified as the most synthesis-feasible design. At the same time, the delta-prediction proposal is applied to the IONIC-SUM architecture as well, to quickly compare these findings with the results from Section 4.2.2 after the LUT-compression optimisation.

4.3.1. Optimisation 2: Design 4 [HYBRID]

The derivative-based reformulation is first applied to the HYBRID architecture. The conductance calculations remain unchanged because they do not rely on derivatives of the original equations. However, the gate-channel LUTs are modified. Instead of predicting next-state variables directly, each LUT predicts the corresponding derivative change of the gating variables, after which Euler propagation is applied to reconstruct the updated state.

Consequently, the gate mappings change from

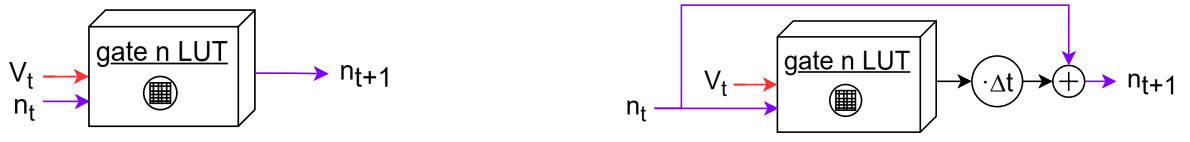
$$(V, n) \rightarrow n_{t+1}$$

to

$$(V, n) \rightarrow \frac{dn}{dt}, \text{ and: } n_{t+1} = n_t + \frac{dn}{dt}\Delta t,$$

with equivalent reformulations for the 'm' and 'h' channels.

As a consequence, the block diagram structure changes from the structure shown in Figure 4.3a to the reformulated structure in Figure 4.3b.



(a) Before Optimisation 2 **(b)** After Optimisation 2

Figure 4.3: Next-state mapping of the gate 'n' channel before and after delta-prediction optimisation (2)

Architecture-wise, this reformulation is applied to every gate-channel LUT, making Design 4 change from the block structure in Figure 3.10 to the diagram in Figure 4.4, with all the 1-dimensional LUTs and multiplications for the conductance currents simplified to a 'V_m' calculation block for simplicity.

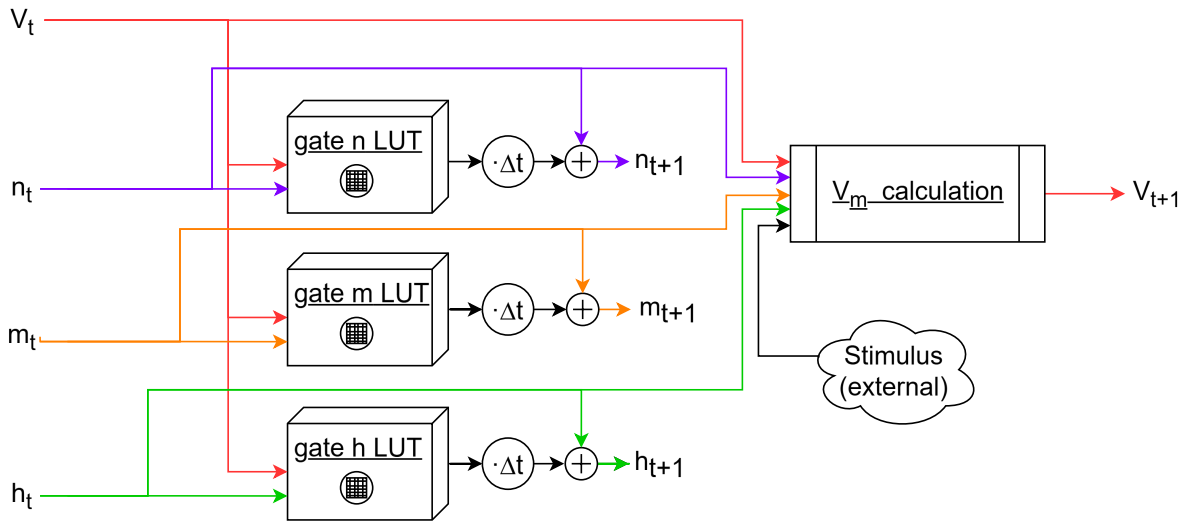


Figure 4.4: Simplified block overview of Design 4 [HYBRID]: reformulated LUTs for the 3 gate blocks implementing delta prediction and Euler propagation, whereafter the voltage is calculated with simple LUTs and arithmetic

Hardware Implications

Introducing Euler propagation adds a small amount of arithmetic to the architecture. Each state now requires multiplication by the time-step parameter Δt, followed by accumulation with the previous state value. Depending on the implementation, these operations can be realised either through DSP blocks or through equivalent shift-add logic.

Since one of the design goals is to minimise DSP utilisation, the multiplication by the fixed timestep is implemented using shift-add arithmetic. The timestep remains constant throughout the simulation and was defined in Section 2.1.1 as $\Delta t = 0.01$ms. Using the 12-bit fixed-point representation of the system, the constant can be approximated as

$$\Delta t \cdot (2^{12} - 1) = 0.01 \cdot 4095 \approx 41,$$

where the constant coefficient can be rewritten as

$$41 = 32 + 8 + 1 = 2^5 + 2^3 + 2^0$$

Because multiplication by a power of two corresponds to a binary left-shift operation, the expression for multiplying some defined variable x by the timestep constant becomes

$$x \cdot 41 = x \cdot (2^5 + 2^3 + 2^0) \tag{4.1}$$
$$= (x \ll 5) + (x \ll 3) + (x \ll 0) \tag{4.2}$$
$$= (x \ll 5) + (x \ll 3) + x \tag{4.3}$$

Consequently, the Euler-propagation stage can be implemented using only shift and addition operations, avoiding the need for dedicated multiplier resources. The additional hardware cost is therefore limited to combinational logic and state-accumulator registers used during state accumulation. The state-accumulator registers are not explicitly shown in Figures 4.3 and 4.4, as the figures abstract away the state-storage infrastructure and focus on the continuous-dynamics datapath. The accumulated 24-bit state is subsequently stored in BRAM within the neuron-core wrapper rather than persistent registers inside the computational core, as discussed in Section 6.2.

The state-accumulators are not shown in Figures 4.3 and 4.4, as they are implemented as BRAM-based state storage within the neuron-core wrapper, outside the continuous-dynamics core. Their implementation is discussed in Section 6.2.

4.3.2. Optimisation 2: Design 3 [IONIC-SUM]

To determine whether the observed behaviour generalises beyond the HYBRID architecture, the same derivative-based reformulation is applied to the IONIC-SUM design. As in Design 4, the gate-channel LUTs predict derivatives of the states, relying on incremental delta changes in the gate-channels. The architecture of the conductance summation remains unchanged, preserving the contributions of potassium, sodium, and leak currents, since it already closely matches the mathematical equations. The visual representation of the transformed design is shown in Figure 4.5.

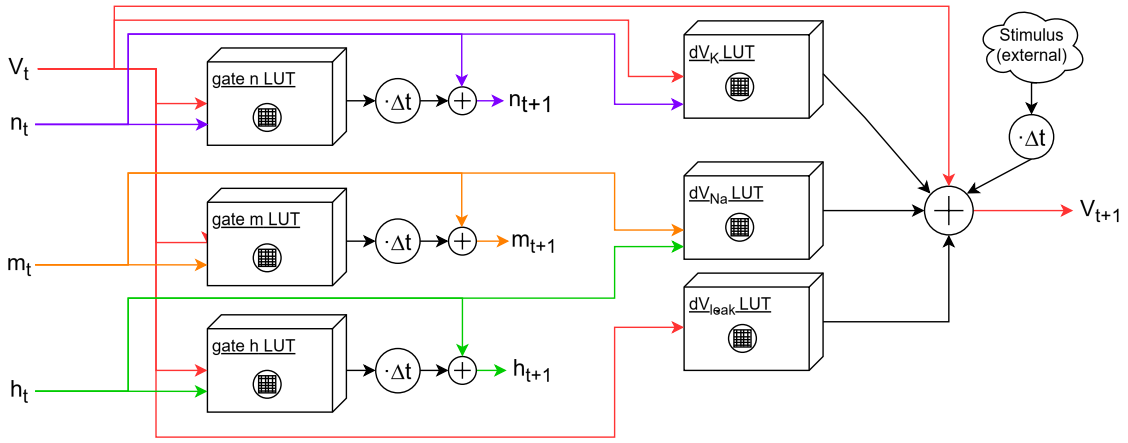


Figure 4.5: Design 3 [IONIC-SUM] with derivative-predicting gate-channel LUTs and Euler-based next state update

Unlike the HYBRID architecture, the IONIC-SUM design still contains higher-dimensional conductance representations, particularly for the Sodium-current dynamics, which may continue to dominate FPGA

resource utilisation despite the improved numerical formulation. Either way, Design 3 provides the opportunity to evaluate whether the effects of derivative prediction remain visible in a more LUT-complex architecture.

4.3.3. Optimisation 2: Requirement Check

Compared to the previous optimisation stage, LUT dimensionality remains unchanged. Since nothing has changed for the number of input combinations or outputs, no reduction in logic utilisation is expected.

As shown before, this reformulation introduced a small amount of additional arithmetic through the Euler update. However, because the multiplication by the fixed timestep can be implemented using shift-add operations, the resulting DSP utilisation remains low and unchanged.

The fixed-point encoding of the timestep introduces a small numerical deviation from the ideal value, as the chosen constant corresponds to $\frac{41}{4095} = 0.010012$ms instead of the intended 0.01ms, introducing a small perturbation of the Euler time grid without affecting the continuous Hodgkin–Huxley dynamics, which are only discretely approximated. The quantisation error corresponds to 0.000012ms per update. Because Euler integration propagates state forward in time, the perturbation is carried through subsequent steps; however, it remains bounded under the observed dynamics due to stable resting behaviour and the transient, non-amplifying nature of spikes. Using a worst-case derivative of 450 mV/ms (see Figure 4.6), the maximum instantaneous voltage deviation is:

$$450 * 0.000012 = 0.0054mV$$

This effect is confined to brief high-activity intervals where dV/dt is large; during resting periods, in which $dV/dt \approx 0$, it vanishes. Overall, the timestep quantisation introduces a small, localised integration perturbation per update that remains bounded and does not alter qualitative spike or equilibrium behaviour.

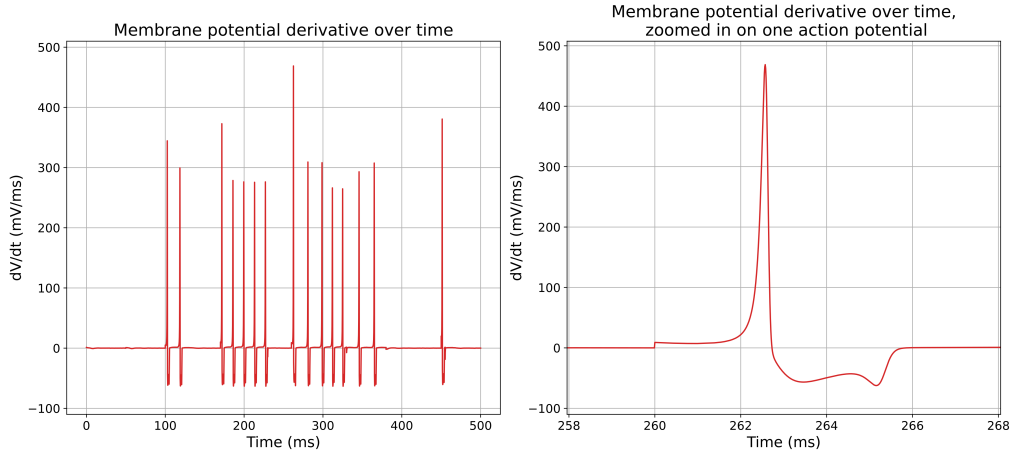


Figure 4.6: Simulated voltage derivatives (dV/dt) over time to visualise the small active regions and the maximum derivative value of around 450mV

An alternative implementation would be to perform the multiplication with dedicated DSP blocks rather than the shift-add approximation, thereby preserving the exact timestep. Assuming the four Hodgkin-Huxley state updates are computed in parallel for time efficiency, this would require one DSP multiplier per state variable, increasing the design from zero to four DSP blocks for the timestep multiplication alone. Although this removes the timestep quantisation error entirely, it introduces a higher computational resource cost. Furthermore, DSP-based multipliers generally exhibit a longer critical path than simple shift-add logic, often requiring additional pipelining to maintain the target clock frequency. While this mainly increases latency rather than reducing throughput in a pipelined implementation, it nevertheless represents an additional hardware trade-off. Since one of the primary objectives of this work

is to minimise computational resource utilisation while maintaining biologically realistic behaviour, the shift-add implementation was selected as the more appropriate design trade-off.

A different approach could be to select a timestep aligned with a power-of-two scaling factor, such as $\frac{1}{128} \approx 0.0078$ms as was implemented in the FPGA architecture designed by Akbarzadeh-Sherbaf et al. [12], which would likewise simplify the hardware by enabling efficient shift-based arithmetic while maintaining comparable numerical accuracy. However, this would require regenerating all datasets to match the adjusted temporal resolution. Given the project constraints and the lack of compelling performance benefit, this modification was not included within the scope of this thesis.

The expected benefit of this optimisation, therefore, lies primarily in improved stability. Since the LUTs now predict local differential behaviour rather than complete future-state mappings, the new recursive system follows the original Hodgkin-Huxley formulation more closely. Since the state update is limited to incremental updates, any prediction error carries less weight than before, so long-term stability is expected to improve.

4.4. Third Optimisation: NeuraLUT

The previous optimisation reformulated the LUT mappings from direct next-state prediction to a delta-based structure consisting of a derivative prediction followed by Euler propagation. While this improved the numerical formulation of the neuron dynamics, the LUTs themselves remained relatively large, even after compression. The next optimisation therefore targets the LUT implementation directly by replacing explicit lookup tables with Neural Network-generated approximations using the adapted NeuraLUT framework introduced in Section 2.3.2 and thoroughly described in Section 5.2.

Unlike the previous optimisation stages, this optimisation is evaluated on both the coarsest-grained and finest-grained architectures. Design 1 (MONOLITH) is included to investigate whether the Neural Network-based representation is sufficient to overcome the dimensionality bottleneck of the monolithic HH implementation. Design 4 (HYBRID) is evaluated because it represents the most practical architecture obtained from the design exploration.

In addition to the standard optimisation chain and evaluation against metrics, a few control experiments are performed. Besides evaluating Neural-Network-generated LUTs after the Delta Prediction optimisation, these LUTs are also generated directly from the next-state prediction model. This allows the influence of derivative prediction on the Neural Network approximation process to be evaluated independently.

At this stage, the mathematical functionality of the neuron model remains unchanged. The modification concerns only the implementation of the LUT mappings. Rather than directly storing all table entries and an output for every input combination, the mappings are first learned by a quantisation-aware trained neural network, after which the network is transformed into a hardware-oriented LUT representation. All perceptrons in this neural network resemble separate LUTs, with weights and activations embedded in their mappings.

4.4.1. Optimisation 3: Design 1 [MONOLITH]

The MONOLITH architecture provides an important benchmark for evaluating whether Neural Network-based LUTs can overcome the dimensionality complexity identified in the design exploration chapter. Whereas the original implementation required a prohibitively large multidimensional lookup table, NeuraLUT replaces the explicit state-transition with a neural network that approximates the same mapping. Now, instead of the complete direct state transition

$$(V_m, n, m, h, I) \rightarrow (V_{next}, n_{next}, m_{next}, h_{next}),$$

the input variables are distributed across several input perceptrons of the neural network, whereafter they are propagated to the subsequent perceptrons in additional layers. After training, the perceptrons of the neuron are converted into a network of low-dimensional LUTs (2 inputs, 1 output) according to the NeuraLUT methodology [17]. Consequently, the implementation no longer scales with the number of entries in the monolithic LUT, but instead with the number of perceptrons needed per layer of the neural network and the bit resolution in each perceptron-derived LUT.

The full updated block diagram is shown in Figure 4.7. Several combinations of network sizes and other parameters were tried; the shown amount in the figure is simply a visual example. The exact simulation details are shown in the simulations and algorithms chapter, Section 5.2.

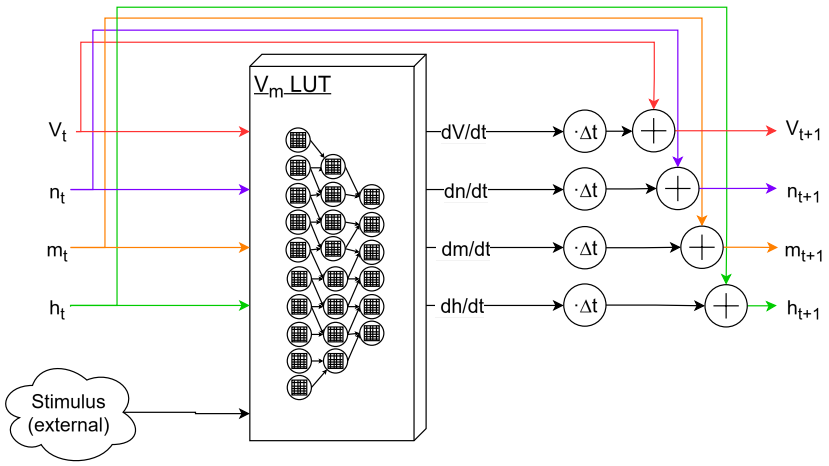


Figure 4.7: Design 1 [MONOLITH]: Neural Network-based LUTs with Euler propagation

4.4.2. Optimisation 3: Design 4 [HYBRID]

The HYBRID architecture represents the primary evaluation platform for this optimisation. In contrast to Design 1, the Hodgkin-Huxley dynamics have already been decomposed into low-dimensional gate-channels and conductance LUTs. Consequently, the neural networks are required to approximate substantially simpler functions. Fewer inputs in the original state-transition LUT means fewer perceptrons are needed in the input layer of the neural network to learn the dynamics of all the input combinations.

For each gate-channel LUT within the architecture, a dedicated neural network is trained using a generated reference dataset. After quantisation-aware training, the networks are transformed into LUT-based hardware structures through the adapted NeuraLUT toolchain, explained in Section 5.3. The resulting implementation consists of interconnected low-dimensional LUTs that represent individual perceptrons in the neural network.

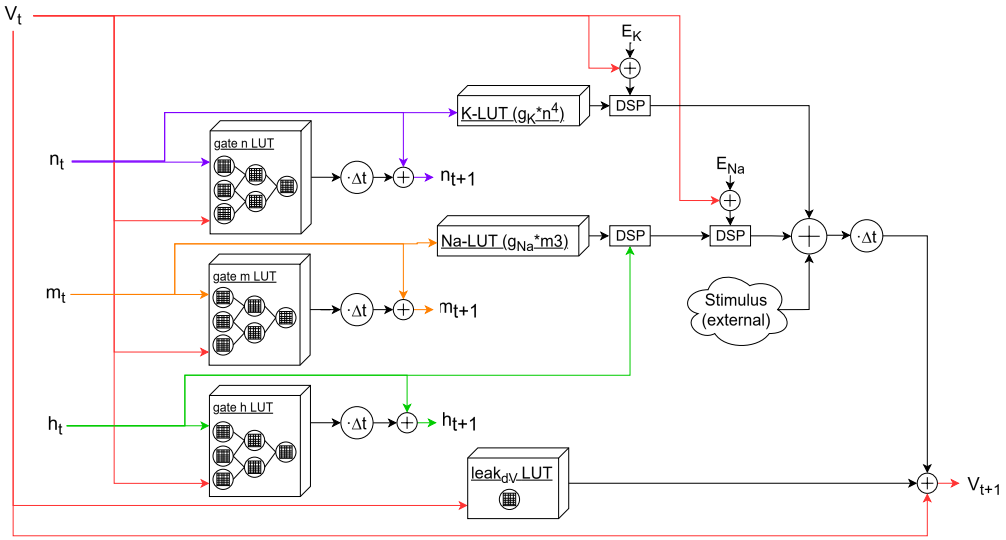


Figure 4.8: Design 4 [HYBRID]: compressed, Neural Network-based gate-channel LUTs with Euler propagation, low-dimensional conductance LUTs and multiplications (DSP)

Because the gate-channel functions are already separated from the membrane-voltage equations, this

architecture enables the effect of this optimisation to be evaluated independently from the dimensionality complexity present in the more coarse-grained designs. Since Designs 2 and 3 share the same gate-channel structures, the results from Design 4 will indicate whether the Neural Network-based approximation strategy is suitable for the other decomposed architectures as well. Additionally, comparison with the control experiment that omits derivative-based predictions allows us to investigate the impact of derivatives, rather than full next states, on Neural-Network approximations.

4.4.3. Control Experiment: Without Delta Prediction

The previous optimisation introduced derivative-based prediction followed by Euler propagation. Consequently, any observed behaviour can originate from either the impact of the delta prediction on the system or the impact of neural networks.

To isolate this effect, additional control experiments are performed in which NeuraLUT is applied directly on the full next-state-prediction LUT. A simplified block diagram example is shown in Figure 4.9.

Figure 4.9: Visual example of the control experiment for optimisation 3 (NeuraLUT) without optimisation 2 (Delta Prediction)

Optimisation 3 (control): Design 4 [HYBRID]

To evaluate the influence of the derivative-based optimisation on the Neural Network, the HYBRID architecture will be used in this configuration. In this configuration, the gate-channel neural networks are trained to predict the complete next-state values directly, rather than the derivatives used throughout the main optimisation strategy.

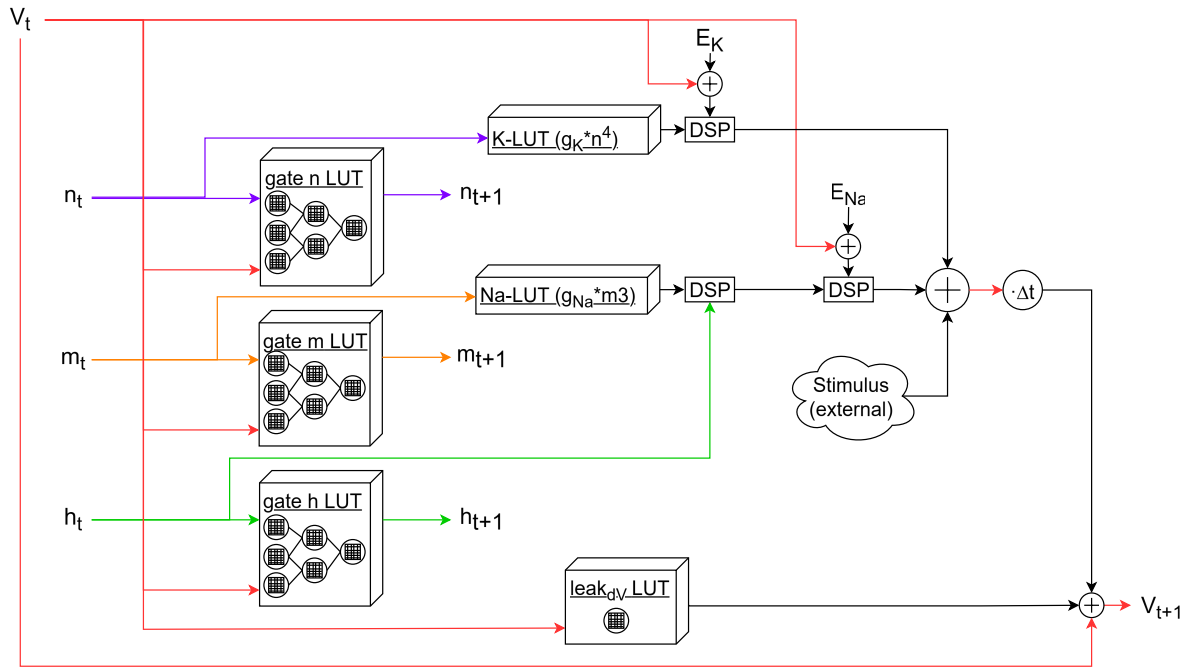


Figure 4.10: Design 4 [HYBRID]: Neural-Network-generated gate-channel LUTs without Delta Prediction

Optimisation 3 (control): Design 1 [MONOLITH]
A similar control experiment is performed for the MONOLITH architecture. The original motivation for introducing Delta Prediction was to provide a more suitable prediction target for these LUT-based systems. This implementation provides a direct comparison between the existing optimisations.

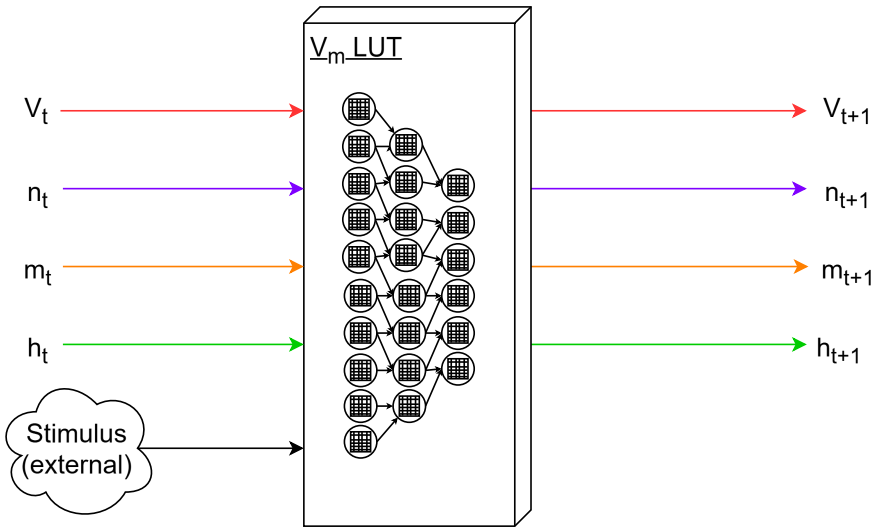


Figure 4.11: Design 1 [MONOLITH]: Neural-Network-generated monolithic state-transition mapping without Delta Prediction

4.4.4. Optimisation 3: Requirement Check

NeuraLUT fundamentally changes the contents of the LUT structures used in all of the designs. Rather than storing explicitly generated mappings, the LUT behaviour is now represented through a network of perceptrons, which is subsequently converted into a hardware-oriented LUT implementation. This means, however, that the dimensionality of resource usage is largely determined by the neural network's size and precision at present.

From a stability perspective, this optimisation has some risks. The optimisation introduces an additional approximation stage because each mapping must now first be learned by the neural network before being converted into LUT form. And, because these neural networks have many parameters that must be carefully chosen and tuned to find the ideal training setup, overall numerical stability decreases.

Logic utilisation is expected to reflect a trade-off between two opposing effects. On the one hand, the original LUT dimensionality is no longer explicitly present. On the other hand, every perceptron of the neural network is converted to a low-dimensional LUT. This introduces a network of LUTs just for 1 variable. However, even with many low-dimensional LUTs, utilisation doesn't return to the same high level as before compression; it settles at roughly mid-high, as will be seen in Section 7.2.3.

DSP utilisation remains unaffected, since the implementation continues to rely solely on LUT-based computation. The additional complexity introduced by the neural network is handled on the computer side during simulation and training, and then incorporated into the LUT structures.

The results of these optimisations determine whether the Neural Network-derived approach provides a favourable trade-off between numerical stability and hardware complexity, thereby motivating the final optimisation stage. However, the introduction of Neural Networks alone is not considered sufficient to significantly improve the evaluated metrics. Instead, its primary benefit lies in enabling further reductions in bit resolution at selected locations within the architecture, allowing additional resource optimisations. This resolution-reduction optimisation is discussed in the following chapter.

4.5. Fourth Optimisation: Bit-Resolution Reduction

The previous optimisation replaced explicit lookup tables with Neural Network-generated LUT structures. Although this significantly reduced dependence on large multidimensional LUTs, it introduced

a new bottleneck. Each perceptron in the generated neural network is ultimately converted into a 2-dimensional LUT, so with a network of perceptrons, which is converted to a network of LUTs, the total logic utilisation depends strongly on the bit resolution used throughout the network.

The original Hodgkin-Huxley implementations adopted a 12-bit representation because the model's dynamics include rapid state transitions during spikes and slow subthreshold variations, all of which must be resolved. However, once these dynamics are distributed across learned functions, weights and activations of the distributed perceptrons, it is no longer evident that every intermediate neural network signal requires the same numerical precision as the final neuron state. Consequently, the final optimisation investigates whether the input- and hidden-layer precisions of the LUT networks can be reduced while preserving a 12-bit output resolution, ensuring the voltage calculation remains accurate. The underlying hypothesis is that the neural network can compensate for the reduced internal resolution through its learned weights and activations, while the high-resolution output ensures sufficient accuracy in the resulting neuron dynamics.

At this stage, Design 1 (MONOLITH) is no longer considered, since this architecture represents the most difficult function-approximation problem and showed limited suitability for Neural-Network-generated LUTs during the previous optimisation stage, due to the significantly greater need for perceptrons and layers than for low-dimensional LUTs. Reducing the internal precision further would not provide additional insight into the architectural trade-offs. Instead, the optimisation is evaluated on more fine-grained, decomposed architectures, in which the neural networks are required to learn lower-dimensional functions. These will establish more realistic candidates for the final implementation and usability in a network of neuron cores.

The modified gate-channel LUTs use reduced-resolution representations for their input and hidden layers of the embedded neural network. In the implemented configurations, the gate-channel networks are tested with 6-bit inputs and 6-bit hidden-layer representations, producing 12-bit outputs that remain compatible with the rest of each architecture in which they are used. The more complex LUTs in Designs 2 and 3 have additional considerations, which will be discussed in the sections below.

4.5.1. Optimisation 4: Design 4 [HYBRID]

The HYBRID architecture remains the primary evaluation platform for the final optimisation due to its lowest complexity. Since the gate-channel dynamics are already separated from the voltage calculations, the effect of reducing neural network precision can be evaluated directly without the additional complexity of large-conductance mappings.

Compared to the previous optimisation stage, the overall architecture remains unchanged, so the block diagram remains as in Figure 4.8. The modification is restricted to the perceptron-based LUTs, in which the embedded input and hidden-layer precisions are altered before conversion to a lookup table. Since the size of LUTs scales exponentially with their input precision, even modest reductions in bit width can substantially reduce the resulting logic requirements. For example, a reduction from two 12-bit inputs to two 6-bit inputs will mean a difference of

$$2^{24} \approx 16777 \cdot 10^3 \text{ bits}$$
$$\text{versus}$$
$$2^{12} \approx 4.1 \cdot 10^3 \text{ bits},$$

which will compensate for the increase in the number of LUTs that results from replacing single LUTs with a LUT network.

4.5.2. Optimisation 4: Design 3 [IONIC-SUM]

Since Design 3 shares the same gate-channel structures as Design 4, it provides a natural secondary evaluation platform for reduced-precision Neural Network-generated LUTs. The gate-channel networks are modified as in the HYBRID architecture, while the conductance-current LUTs are modified with a more conservative reduction in resolution. Unlike the gate-channel dynamics, the conductance-current LUTs couldn't directly benefit from the derivative-based reformulation introduced by Optimisation 2.

Consequently, these LUTs must still represent a wide numerical range that covers both slow and rapid changes in ionic current contributions. Excessive bit-width reduction may therefore introduce substantial approximation errors. For this reason, an appropriate trade-off is investigated between internal bit resolution within each LUT and the neural network complexity, the balance between the number of perceptrons and the amount of hidden layers used in the NeuraLUT implementation.

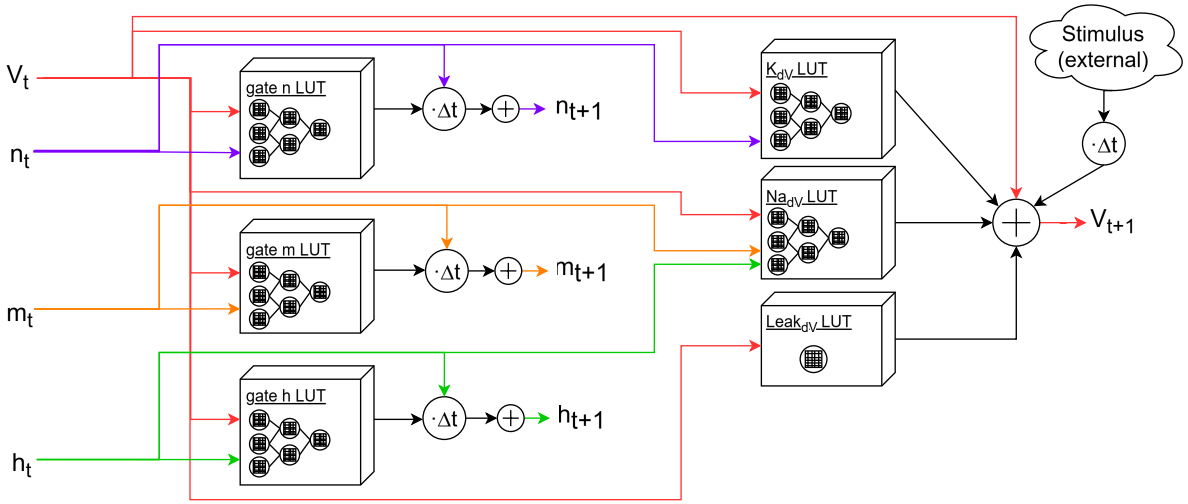


Figure 4.12: Design 3 [IONIC-SUM]: reduced-precision Neural Network-generated gate-channel and conductance-current LUTs, with the leak-current contribution as a direct 1-dimensional LUT

Furthermore, the DSP-less, modular structure of the IONIC-SUM architecture remains attractive for future extensions of the Hodgkin-Huxley model that involve additional ion-channel dynamics, such as calcium-based (Ca^{+}) ions, as briefly discussed in Chapter 2. This provides additional motivation to investigate whether reduced-precision Neural Network-generated LUTs remain compatible with the decomposed ionic-currents architecture.

Architecturally, the modifications are similar to those applied in Design 4. The original low and multi-dimensional LUTs are replaced by neural networks, which are subsequently transformed into networks of low-dimensional LUTs. The resulting architecture is illustrated in Figure 4.12. Since the leak-current LUT already consists of a simple 1-dimensional mapping, it is kept in its original form, as no meaningful reduction in LUT complexity and logic usage is expected from replacing it with a neural-network representation.

4.5.3. Optimisation 4: Design 2 [SPLIT]

Unlike Designs 3 and 4, the SPLIT architecture was not evaluated during the previous optimisation stages. However, the reduced-precision optimisation provides an opportunity to investigate whether the remaining high-dimensional voltage-membrane LUT can be approximated accurately by a neural network while simultaneously reducing the resulting hardware cost through a lower internal bit resolution.

Compared with the MONOLITH architecture in Design 1, the voltage LUT in Design 2 represents a substantially simpler function because it only predicts the membrane-voltage update, using all states as inputs to aid the prediction. Nevertheless, the voltage LUT remains a complex component because more input states require additional input-layer perceptrons to capture all possible combinations.

The success of this experiment is dependent on two results. First, the results must demonstrate that Neural Network-generated LUTs can learn the relatively high-dimensional voltage mapping. Second, they have to show that the reduced resolution can sufficiently reduce the resulting total logic usage to make the architecture practical for synthesis and implementation. Since the expectation is that, to achieve successful results for the first objective, a relatively higher number of perceptrons and hidden layers are needed to learn the high voltage-membrane complexity, it will be challenging to realise the

second objective.

As in the previous architectures, the original LUT blocks are replaced by a network of low-dimensional LUTs in which a neural network is embedded. The gate-channel networks use the same reduced-precision configuration as Designs 3 and 4, while the voltage-membrane is assigned a more conservative configuration due to its increased functional complexity.

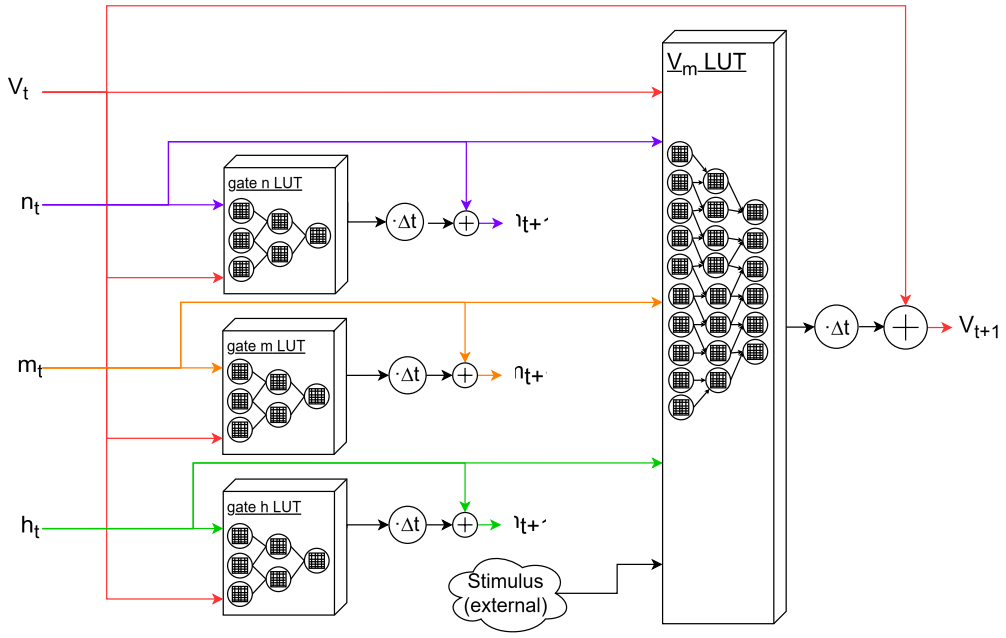


Figure 4.13: Design 2 [SPLIT]: reduced-precision Neural Network-generated gate-channel LUTs together with a Neural Network-generated voltage-membrane LUT replacing the original high-dimensional mapping

4.5.4. Optimisation 4: Requirement Check

Reducing the internal bit resolution of the Neural Network-generated LUTs is expected to substantially reduce logic utilisation. Since LUT storage scales exponentially with the input bit width, even a modest reduction in resolution can significantly reduce the resulting logic utilisation. Consequently, this optimisation is expected to offer the largest improvement in logic efficiency among the optimisation stages.

The reduction in resolution, however, introduces an additional source of approximation error. Fewer representable states are available in the intermediate LUTs, potentially reducing the networks' ability to accurately reproduce the original dynamics. Therefore, stability is expected to decrease relative to the higher bit resolution implementation in Optimisation 3.

DSP utilisation is expected to remain unchanged. The optimisation only modifies the bit resolution of the Neural Network-generated LUTs and does not introduce any additional arithmetic operations or alter the computational flow.

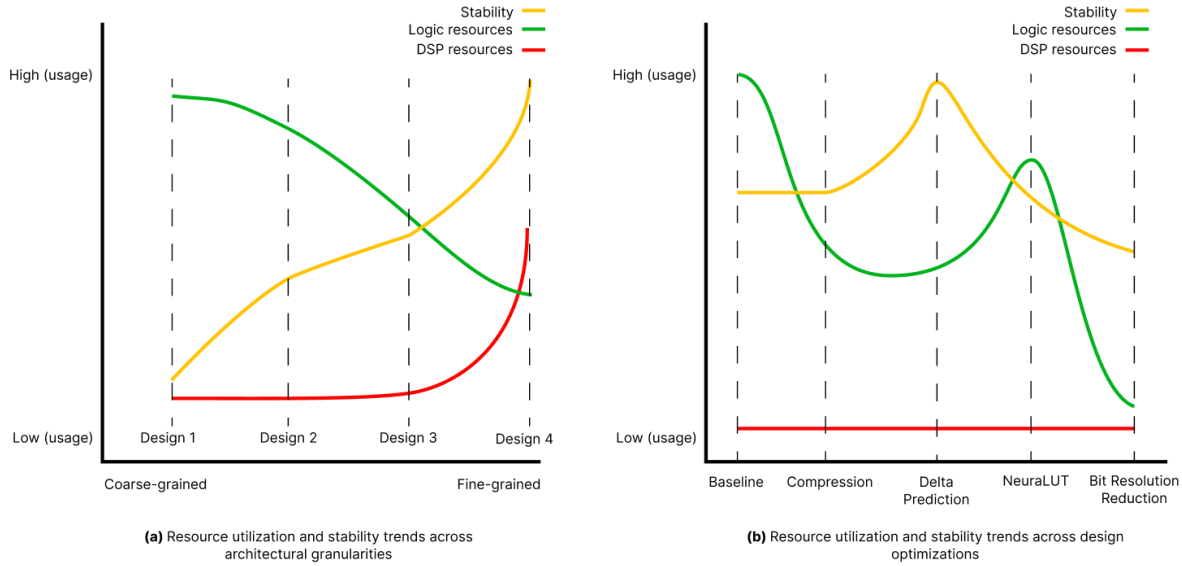
Overall, this optimisation investigates the trade-off between numerical accuracy and hardware efficiency. If the reduction in logic utilisation outweighs the associated loss of stability, the resulting architectures may provide a more practical balance between resource cost and biological accuracy, making them strong candidates for the final implementations evaluated in this thesis and for implementing a neuronal network within the neuron-core wrapper design.

4.6. Optimisation Evaluation

The implementation chapter explored a sequence of optimisation strategies to transform the original LUT-based Hodgkin-Huxley architectures into feasible hardware implementations that satisfy the Neuron Core design requirements. The objective was to evaluate how successive optimisations affect

hardware feasibility and metric evaluation when applied cumulatively, using Design 4 (HYBRID) as the primary reference and Designs 1-3 as supporting cases where applicable.

The evaluations visualised in Figure 4.14 combine two perspectives: a recap of the comparison across architectural designs and the stepwise evolution of the baseline system through the full optimisation chain, up to the final reduced-resolution configuration. Together, these views provide a consistent representation of how the system properties change under progressive design modifications.



(a) Resource utilization and stability trends across architectural granularities

(b) Resource utilization and stability trends across design optimizations

Figure 4.14: Evaluation of the design architectures (a) and of the optimisations (b) across the Neuron Core metrics. Optimisations are evaluated primarily based on Design 4 [HYBRID].

Across this trajectory, the following trends are observed with respect to the neuron-core metrics:

Long-term Stability

A clear trade-off was evident for the stability metric. The derivative-based reformulation, together with the Euler propagation step (Optimisation 2), is designed to dramatically improve effective resolution and subthreshold simulation accuracy. On the other hand, the introduction of learned neural network functions (Optimisation 3) and related precision reduction (Optimisation 4) introduces compounding approximations that are expected to challenge the long-term stability.

Efficient Logic-Block Utilisation

While algorithmic LUT compression (Optimisation 1) will greatly reduce the logic usage of LUT blocks, the bit-resolution reduction in the Neural Network-derived LUTs (Optimisations 3+4) is also expected to yield a significant reduction in physical FPGA logic.

Reduced DSP-Block Utilisation

Throughout the entire trajectory, DSP block utilisation is successfully held to an absolute minimum (Design 4) or eliminated within the structures (Designs 1 to 3). This is achieved by mapping the explicit timestep scalar introduced in Optimisation 2 to an exact, multiplierless shift-add hardware approximation.

Ultimately, this structural progression demonstrates that overcoming the exponential scaling of biological neuron models requires a hybrid optimisation strategy that combines mathematical reformulations with hardware-aware precision tuning. The requirement checks presented in this chapter represent design hypotheses and expected trade-offs based on architectural analysis. Their actual impact on simulation accuracy and FPGA resource utilisation is confirmed through the simulation results and synthesis statistics presented in detail in Chapter 7.

5

Simulation Framework and Algorithm Optimisation

This chapter describes the complete workflow for transforming the Hodgkin–Huxley neuron model into an FPGA implementation, specifically conforming the designs and optimisations from Chapters 3 and 4. Beginning with numerical simulation of the Hodgkin–Huxley equations, the chapter then follows the successive stages of dataset generation, neural network training, LUT conversion, logic compression, and hardware integration. The overall design flow is summarised in Figure 5.1, and each framework is discussed in detail thereafter. Specifically, the following simulation and algorithm frameworks will be discussed, including short explanations and applied modifications, if applicable:

- Python: HH-model simulation for training and validation
- Brevitas: Training of a Neural Network to predict HH with Quantisation-Aware Training (QAT) mode
- NeuraLUT: Conversion of Neural Network to truth tables in hex and binary format for Verilog use
- ReducedLUT: Compression of LUTs with subtable matching algorithm, from hex-format truth tables to Verilog files
- Verilog: Integration of the separate LUTs to form the Neuron Core pipeline and later the neuronal network

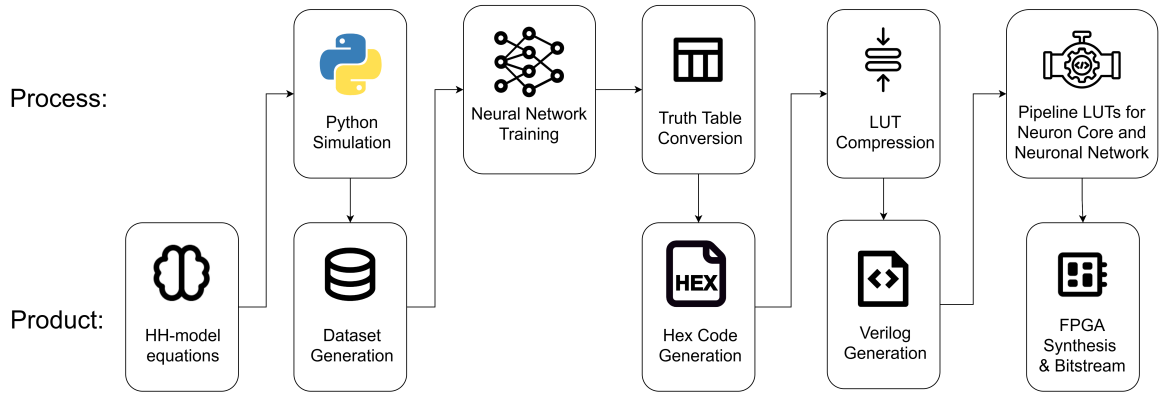


Figure 5.1: Visualisation of the full workflow designed in this thesis, starting from HH equations to the synthesis of the neuron core and network on an FPGA. The top row visualises the processes, and the bottom row shows the products from this that are used in the next framework.

5.1. Python: HH Simulation

A software implementation of the Hodgkin-Huxley (HH) neuron model was developed in Python to generate reference data for neural network training and hardware validation. The model was simulated using forward Euler integration with a timestep of $\Delta t = 0.01ms$.

The implemented parameterisation follows the reference Python HH implementation from the OpenWorm Hodgkin-Huxley tutorial [60] and is shown in Table 5.1.

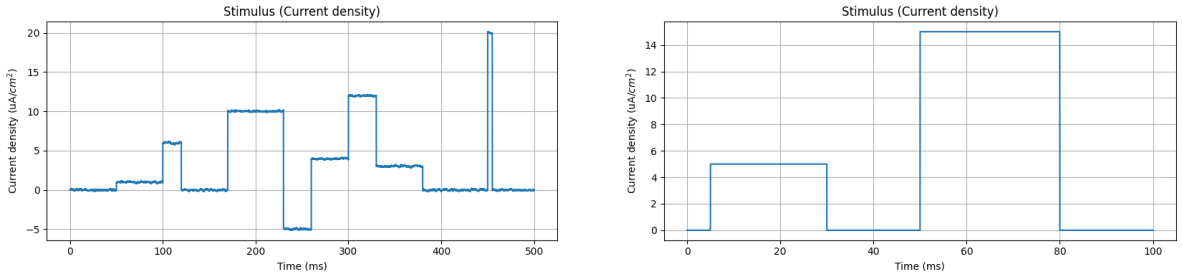
Parameter	Value	Function
Δt	0.01 ms	Simulation timestep
C_m	1.0 μF/cm^2	Membrane capacitance
g_{Na}	120 mS/cm^2	Sodium conductance
g_K	36 mS/cm^2	Potassium conductance
g_L	0.3 mS/cm^2	Leak conductance
E_{Na}	63.54 mV	Sodium reversal potential
E_K	-74.16 mV	Potassium reversal potential
E_L	-54.3 mV	Leak reversal potential
V_0	-65 mV	Initial membrane potential

Table 5.1: Hodgkin-Huxley simulation parameters used for all simulations

5.1.1. Dataset Generation

The training dataset was generated from a 500 ms simulation using a varying stimulus input covering multiple neuronal operating regimes, including subthreshold behaviour, repetitive firing, inhibition, rebound activity, and transient high-current excitation. The applied stimulus is shown in Figure 5.2a.

To improve robustness against small input variations, bounded Ornstein-Uhlenbeck noise was added to the stimulus current. The noise process used a time constant of $\tau = 1$ ms and standard deviation $\sigma = 0.1$, with the noise contribution clipped to the range $[-0.25, 0.25]$ μA/cm^2.



(a) Training stimulus profile.

(b) Validation stimulus profile.

Figure 5.2: Stimulus currents used for HH simulation

Separate validation simulations used constant current stimuli that were not included during training. Currents of 5 and 15 μA/cm^2 were selected to evaluate moderate and strong excitation conditions, respectively. These cases were later reused to compare the software model, neural-network approximation, and FPGA implementation. The validation stimulus is shown in Figure 5.2b. The intermediate gating variable, conductance current and the resulting voltage values from these stimuli can be found in the figures in Appendix Section B.1.

5.2. Brevitas: Neural Network Training

Before converting the neural network into FPGA-compatible lookup tables, the model is trained on the aforementioned dataset with quantisation constraints in mind. Standard floating-point neural networks are not directly suitable for LUT-based hardware implementations, as their weights and activations require higher precision, thereby requiring larger memory resources. Therefore, Quantisation-Aware Training (QAT) is used to train the network, along with tools that account for the limited numerical precision of the hardware representation, such as dataset normalisation prior to training.

5.2.1. Normalisation

Prior to training, all network inputs and target values are normalised to fixed ranges that match the intended hardware representation. This improves training stability, ensures that all network signals occupy a similar numerical range, and simplifies the subsequent quantisation process.

The membrane voltage is bounded by the biologically relevant range of $[-128, 128]$ mV and linearly mapped to the interval $[-1, 1]$ according to

$$V_n = 2 \cdot \frac{V - V_{\min}}{V_{\max} - V_{\min}} - 1, \tag{5.1}$$

where $V_{\min} = -128$ and $V_{\max} = 128$.

Similarly, the gating variables m, n, and h are naturally constrained to the interval $[0, 1]$. These values are linearly mapped to a zero-centred interval $[-1, 1]$ using

$$x_n = 2x - 1, \tag{5.2}$$

where $x \in m, n, h$. The neural networks are therefore trained entirely in the normalised domain. Both the network inputs and the predicted outputs are represented within these bounded ranges. This allows the quantisation layers to operate on a fixed numerical interval independent of the physical units of the underlying Hodgkin–Huxley variables.

After inference, the normalised outputs can be converted back to their physical representations using simple scaling operations. For the gating variables, the inverse transformation maps $[-1, 1]$ back to $[0, 1]$. For the membrane voltage, the inverse transformation scales the normalised value to the interval $[-128, 128]$ mV. Because these ranges were intentionally selected as powers of two, the corresponding hardware implementation can be realised efficiently using fixed-point shifts and additions rather than general multiplication operations.

5.2.2. Quantisation-Aware Training (QAT)

QAT simulates the effect of fixed-point quantisation during training by applying the same reduced-precision representation used during FPGA inference. This allows the network weights and activation ranges to adapt to quantisation error before deployment. Training was performed using the QuantBrevitas library, which provides quantisation-aware PyTorch layers.

A specific constraint arises from the autoregressive nature of the prediction: the network's quantised output at timestep t is fed back directly as its input at timestep $t+1$. If the input and output quantisation scales were allowed to differ — or to change between training epochs — a systematic numerical offset would accumulate across timesteps and eventually corrupt the simulation. To prevent this, fixed scaling factors are enforced to the input and output of the network via `ScalingImplType.CONST`, rather than learned or statistics-based scales.

Listing 5.1 shows the Brevitas function that creates a fixed-scale quantiser for a given bit-width. The quantiser is constructed through Python's `type()` call, which dynamically creates a new class that inherits from Brevitas's `Int8ActPerTensorFloat` base and overrides the relevant configuration fields.

Listing 5.1: Fixed-scale quantiser factory for autoregressive-compatible input/output quantisation.

```python
def make_fixed_scale_quant(bit_width: int):
    return type(
        f"FixedScale{bit_width}Bit",
        (Int8ActPerTensorFloat,),
        {
            "bit_width": bit_width,
            "quant_type": QuantType.INT,
            "scaling_init": 1.0,
            "scaling_impl_type": ScalingImplType.CONST,
            "restrict_scaling_type": RestrictValueType.FP,
            "return_quant_tensor": False,
        },
    )

# Applied to the network layers as:
QuantIdentity(act_quant=make_fixed_scale_quant(model_config["input_bitwidth"]))
```

Hidden-layer activations use `QuantHardTanh` with a learnable scale (`ScalingImplType.PARAMETER`) and a symmetric clipping range of $[-1, 1]$. This provides flexibility during training while keeping intermediate values bounded and compatible with the fixed-point format expected by the LUT conversion step. Intermediate values between the perceptrons are not used for anything, hence they can be dynamically scaled to optimal ranges for the perceptrons, as long as each hidden layer perceptron's output matches the scale of the input of the connecting perceptron.

5.2.3. Network Architecture and Forward Propagation

MLP Structure

The HH simulation is approximated by one or more Multi-Layer Perceptron (MLP) networks, each predicting a subset of the state variables. Splitting the prediction across several smaller networks reduces the per-LUT complexity after conversion, compared to a single large network, as discussed in Section 3.2. The network geometries were selected empirically with the objective of minimising hardware cost while maintaining sufficient prediction accuracy during autoregressive testing, in which the output state is used directly as input to the next state update. Initial architectures were chosen based on the number of network inputs and outputs, after which hidden-layer sizes were increased only when smaller configurations no longer achieved the required accuracy. This process resulted in a $(4, 2)$ architecture for the individual gate-variable and ionic-current predictors, while the voltage predictor required a larger $(10, 8, 6)$ configuration to model the more complex dependency on all five state variables.

Fan-in constraints and the connectivity mask

As discussed in the background, NeuraLUT imposes a limited fan-in F: each output neuron may connect to at most F inputs from the preceding layer. This keeps the truth table size bounded at $2^{F \cdot b_{\text{in}}}$ entries, where b_{in} is the input bit-width. The connectivity is defined by a fixed mask $M \in \mathbb{Z}^{N \cdot F}$, where row M_n lists the F input indices wired to output neuron n.

Choosing which F connections to keep matters: training directly on a randomly wired sparse network is difficult because important input relationships may be ignored before the network has any chance to learn them. To address this, a dense-to-sparse training scheme adapted from [61] is used, as illustrated in Figure 5.3.

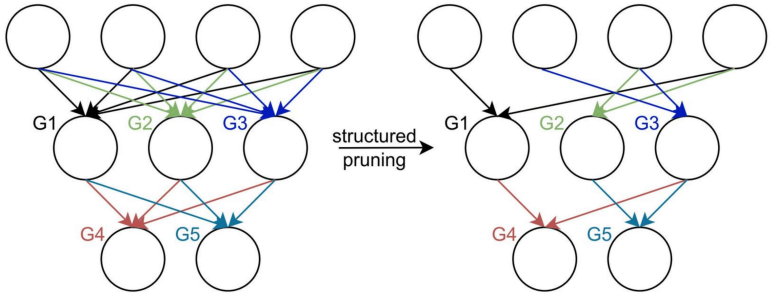


Figure 5.3: Visualisation of the transformation from a dense fully-connected network to a sparse, pruned-connectivity network, replicated from Figure 5 in [61]

Phase 1: Dense pre-training
The network is first trained with full connectivity (`dense_forward = True`), so every neuron can reach every input. After a short pre-training run, the weight matrix of the linear residual path $W_r \in \mathbb{R}^{N \cdot N_{\text{in}}}$ is inspected. Because W_r is a direct linear mapping from all inputs to each output neuron, the magnitude $|W_{r,ni}|$ indicates how strongly input i contributes to neuron n. For each perceptron row, the F indices with the largest absolute weight are selected for further use. This produces the connectivity mask M that reflects the input dependencies the network actually learned, rather than an arbitrary wiring.

Phase 2: Sparse fine-tuning
The mask M is fixed and `dense_forward` is set to False, switching the network to its hardware-compatible sparse structure. The network is retrained from the dense checkpoint for significantly more epochs, allowing the weights to adapt to the constrained connectivity. Because the mask preserves the most informative connections, fine-tuning only needs to compensate for the discarded ones, and the final network exactly matches the eventual FPGA wiring.

The **`SparseLinearNeq`** layer

The core trainable building block is `SparseLinearNeq` (Sparse Linear Neuron Equation). Recall from the background that a standard perceptron computes:

$$y = \sigma\left(\sum_{i=1}^{N_{\text{in}}} w_i\, x_i + b\right), \tag{5.3}$$

i.e. a single weighted sum of all N_{in} inputs followed by a nonlinear activation σ.

`SparseLinearNeq` replaces the single weighted sum of the standard perceptron with a small two-layer sub-MLP per output neuron, augmented by a linear skip connection. A visual representation of the adapted structure of the perceptron is shown in Figure 5.4.

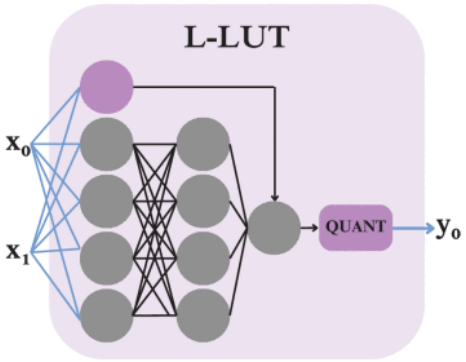


Figure 5.4: Visual representation of the hidden sub-MLP structure within a perceptron, rebranding it to an 'L-LUT', adapted from Figure 4 in [17]

Let $X \in \mathbb{R}^{N_{\text{in}}}$ denote the full quantised input vector to the layer. The computation for output neuron n proceeds in three steps:

Step 1 - input selection:
Rather than using all N_{in} inputs, neuron n receives only the F values selected by its mask row M_n:

$$\tilde{x}_n = X[M_n] \in \mathbb{R}^F. \tag{5.4}$$

Step 2 - sub-MLP:
The selected inputs pass through a small two-layer network. A first linear layer expands them to N_w hidden activations h_n, in where ReLU introduces nonlinearity, and a second linear layer compresses the result back to a single scalar z_n. This means the perceptron can represent significantly more complex nonlinear functions than a single weighted sum. This is the key reason NeuraLUT achieves higher accuracy for a given LUT size compared to simpler approaches.

$$h_n = \text{ReLU}(W_1 \tilde{x}_n + b_1), \qquad \text{where } W_1 \in \mathbb{R}^{N_w \cdot F}, \tag{5.5}$$

$$z_n = W_2 h_n + b_2, \qquad \text{where } W_2 \in \mathbb{R}^{1 \cdot N_w}. \tag{5.6}$$

The hyperparameter N_w (`width_n`) controls how many hidden units the sub-network uses, and therefore how expressive each neuron is.
A value of $N_w = 12$ was used for all experiments without further sweeping. This is a deliberate but conservative choice: with only $F = 2$ inputs per perceptron, each sub-network approximates a low-dimensional slice of the HH dynamics, and there is little reason to expect that $N_w = 64$ or 128, which is often more common in larger architectures, would yield meaningful improvement. At the same time, twelve hidden units provide noticeably more capacity than a purely linear mapping and are sufficient to capture the present nonlinear structure, while remaining proportional to the overall network scale.

Step 3 - residual and quantisation:
A separate linear mapping r_n takes the same F inputs and adds them directly to the sub-MLP output before quantisation:

$$r_n = W_r \tilde{x}_n + b_r, \qquad \text{where } W_r \in \mathbb{R}^{1 \cdot F}, \tag{5.7}$$

$$y_n = \text{Quantize}(z_n + r_n). \tag{5.8}$$

Practically, such a skip connection provides a shortcut for gradient flow during backpropagation, which stabilises training, and allows the nonlinear path to focus on learning the deviation from a simple linear response rather than the full function [62]. Moreover, the residual adds no hardware cost: since the entire sub-network is compiled into a single LUT after training by precomputing all input-output combinations for $2^{F \cdot b_{\text{in}}}$ possible inputs, the residual is simply absorbed into the lookup table alongside the nonlinear path, with no additional FPGA logic required [17].

The full network is obtained by stacking multiple `SparseLinearNeq` layers, with all N neurons within a layer computed in parallel. Each layer's output quantisation range matches the next layer's input range exactly, so quantised values propagate without rescaling. After training, one truth table is generated per neuron per layer and synthesised as a LUT on the FPGA.

5.2.4. Training Methodology and Parameter Tuning

Training low-precision sparse networks to model complex differential equations requires specialised hyperparameter tuning and distinct training phases.

Training a low-precision sparse network to reproduce complex differential equation dynamics requires careful design of the loss function, multi-step autoregression strategy, and optimisation schedule. Following the dense-to-sparse connectivity mask generation in the previous chapter, the decisions made during those training processes are detailed below.

Loss Functions and Velocity Weighting

Since the HH neuron remains close to the resting state for most of the simulation, a standard loss function tends to be dominated by these low-dynamic regions. This can reduce the accuracy during

short but important changing events, such as action potentials. For example, the widely used Mean-Square-Error (MSE) loss function squares each error, so a prediction error of 10 contributes 100 to the loss while an error of 1 contributes only 1. During action potentials, where the gate variables change rapidly, prediction errors are inevitably large, and squaring them causes those few timesteps to dominate the total loss almost entirely. As an alternative, the SmoothL1 Loss function behaves quadratically for small errors but switches to a linear penalty above a threshold, so large spike-region errors still contribute but no longer overwhelm the gradient signal from the much more numerous resting-state samples.

This is sufficient for the gate-channel dynamics; however, the ionic-currents exhibit even more rapid variations and therefore require additional measures. On top of the Smooth1L loss function, a velocity-based weighting scheme is applied to compensate for the imbalance between long resting periods and short spiking events. For each sample, the rate of change $\dot{g} = |g_{t+1} - g_t|$ is used as a measure of local dynamic activity. To make this measure independent of the absolute scale of the gating variable, it is normalised by the maximum observed velocity. The resulting value is used in an exponential decay weighting function:

$$w = 1 + 9 \cdot \exp\left(-5 \, \frac{\dot{g}}{\max(\dot{g})}\right). \tag{5.9}$$

The exponential term provides a smooth transition between slowly varying and rapidly changing regions. The decay factor 5 controls the sharpness of this transition and was selected empirically; smaller or larger values resulted in insufficient functionality of this decay. The coefficients 1 and 9 define the weighting range, resulting in weights between approximately 1 and 10. Consequently, low-velocity (resting) samples receive a weight close to 10, whereas high-velocity (spiking) samples remain close to the base loss contribution. This prioritisation is motivated by the model's autoregressive nature: small prediction errors during long quiescent periods can accumulate over time, introducing drift. Increasing the contribution of resting-state samples, therefore, encourages accurate long-term state propagation rather than optimising only for short action-potential transitions.

Multistep Rollout

Autoregressive Rollout Training

The common single-step training of neural networks is insufficient for our implementation, since during hardware inference, the network feeds its own output back as the next input, so any tiny prediction error compounds heavily over time. The training loop therefore rolls out the network for L consecutive steps, accumulates the weighted loss across all steps, and performs a single backward pass. Two complementary strategies ensure training remains stable as the rollout grows.

Progressive rollout

Rather than training on long sequences from the start, the rollout length L increases linearly with the epoch progress:

$$L(epoch) = \min(2 + \lfloor epoch/25 \rfloor \, , \; L_{\max}) \, . \tag{5.10}$$

Early epochs use very short rollouts of a few steps, where gradients are well-behaved and the network can learn basic one-step accuracy. The horizon grows gradually as the network improves, eventually reaching sequences of hundreds of steps. Gradient norms are clipped to 1.0 throughout to prevent instability during longer rollouts.

Teacher forcing

During early epochs, the true ground-truth state from the dataset is used as input for that timestep. This is done with a probability τ that decays over training:

$$\tau(epoch) = \max\left(0.05, \; 0.8 \cdot \left(1 - \frac{epoch}{\text{total epochs}}\right)\right). \tag{5.11}$$

Without this, the network would be trained on its own compounding errors before it could make reasonable predictions, causing early drift. As training progresses, τ decays toward zero, gradually transitioning the network to a fully autonomous rollout.

Algorithm 1 summarises the combined procedure.

Algorithm 1 Progressive autoregressive training

Precondition: Trajectory X, total epochs E, maximum rollout $L_{\max}$

for each epoch e **do**
  $L \leftarrow \min(2 + \lfloor e/25 \rfloor,\ L_{\max})$ ▷ Rollout curriculum
  $\tau \leftarrow \max(0.05,\ 0.8\,(1 - e/E))$ ▷ Teacher forcing ratio
  Sample random start index; set $x_0 = X[\text{start}]$, $\mathcal{L} = 0$
  for $t = 1$ **to** L **do**
    $x_{\text{in}} \leftarrow X[t{-}1]$ with prob. τ, else x_{t-1} ▷ Ground truth or own prediction
    $\hat{x}_t \leftarrow x_{\text{in}} + \text{Network}(x_{\text{in}}) \cdot \Delta t$ ▷ Predict next state
    $\mathcal{L}$ += WeightedSmoothL1$(\hat{x}_t,\ X[t])$
    $x_t \leftarrow \hat{x}_t$
  Backpropagate $\mathcal{L}$; clip gradients (± 1.0); update weights

The training procedure described above applies to the gate variable networks. Ionic-current networks (ΔV_K, ΔV_{Na}) require longer rollouts to cover a full spike-and-recovery cycle, with sequences extending to 200 steps. Backpropagating through sequences of this length in a single pass is both memory-intensive and numerically unstable, so TBPTT (Truncated Backpropagation Through Time [63]) is applied with a window of $K = 20$ steps: gradients are accumulated over 20 consecutive steps, a parameter update is performed, and the computational graph is detached before the next window begins. This caps the effective gradient depth at 20 steps regardless of total rollout length.

The loss weighting is also inverted relative to gate training. Rather than upweighting resting-state samples, spike-region samples receive the higher weight. For ionic-current prediction, the voltage swing during an action potential is the primary modelling challenge and must be reproduced precisely; the flat resting contribution is comparatively simple and does not require extra emphasis. The AdamW optimiser is used with a weight decay of 10^{-5} for regularisation. The learning rate follows a cosine annealing schedule with warm restarts (period $T_0 = 500$ epochs), which periodically resets it to help escape local minima as the rollout horizon extends.

5.2.5. Evaluation: Autoregressive Testing

After training, each network is evaluated in the same autoregressive mode used during hardware simulation: the network's own predicted output is fed back as the next input, without access to ground-truth states at any point. This directly tests whether prediction errors are eliminated during training. Accuracy is evaluated on the original physical values using MSE, RMSE, and R^2, providing a more interpretable measure than the normalised training. Before LUT conversion, the input ranges observed during the autoregressive simulation are recorded for each neuron.

Output quantisation ranges

The output quantisation range differs between network types and must be set correctly after the hardware evaluation with the quantised values. Gate variable networks predict a normalised delta, the change in the gate variable per timestep — rather than the gate value itself, as discussed in Optimisation 2 of Chapter 3. During rapid transitions such as sodium channel activation (m gate), this delta can substantially exceed the $[-1, 1]$ range used for the gate values. Clipping the output in the OutputQuant layer to ± 1 would silently truncate those fast dynamics and corrupt the simulation, so the output quantiser is set to $[-8, 8]$:

```
QuantHardTanh(bit_width=12, min_val=-8.0, max_val=8.0,
              act_scaling_impl=ScalingImplType.CONST)
```

Ionic-current networks use $[-1, 1]$, which is sufficient because their contribution to the voltage step is normalised into that range by construction.

The choice of range has a direct hardware consequence. A LUT stores integer codes, and the scale factor embedded in the quantiser determines what physical value each code represents. For gate networks, the downstream circuit must apply a factor of 8 to the raw LUT output to recover the correct gate delta; for ionic-current networks, no additional scaling due to the neural network is required. However,

they are normalised to ranges other than ± 1, namely ± 16, so they should be denormalised with appropriate shifts before summation to determine the voltage state. All these scaling factors are fixed and known at synthesis time, so they introduce no runtime overhead on the FPGA.

5.3. NeuraLUT: Conversion to Hardware

After training, each quantised neural network is converted into FPGA-compatible hardware language files. This is done by exhaustively looping over the complete quantised input range for each perceptron in the MLP and storing the results as a truth table. Each truth table is translated into a synthesizable, albeit large, monolithic Verilog case statement, which maps directly to logic block resources during FPGA synthesis. Additionally, the truth tables are converted to hex-format output-only text files for use with other tools that work with LUTs, such as ReducedLUT [18].

The number of truth-table entries per perceptron in determined by the fan-in F and the input bit-width B:

$$S = 2^{F \cdot B} \tag{5.12}$$

For the fan-in of $F = 2$ used in this work, increasing the input precision from 6 bits to the required 12 bits per variable increases the per-neuron state space from $2^{12} = 4096$ to $2^{24} = 16\,777\,216$ entries. The original NeuraLUT pipeline [17] was designed for low-resolution networks that use deep, large sub-networks within the perceptrons and cannot handle state spaces of the required resolution without modifications. Original execution at 120bit precision would have required an estimated conversion time on the order of weeks on the available equipment during this project. To this end, three targeted adaptations were made to reduce the runtime to a practical level.

5.3.1. Adaptation 1: Chunked Truth-Table Generation

Original behaviour

The original `calculate_truth_tables` function generated the complete set of S possible input combinations for a perceptron and stored them in GPU memory simultaneously before evaluating the network. For each input combination, the F input values are stored separately, resulting in an input matrix of size $S \cdot F$. Storing the complete matrix at once exceeds the available GPU memory, causing an out-of-memory error on consumer GPUs before the truth-table generation can be completed.

Adaptation

The complete permutation matrix is now in CPU RAM. Only one chunk of $C = 2^{20}$ rows is transferred to GPU memory at a time. Once the network output for that chunk has been computed and moved back to the CPU, the GPU memory is freed before the next chunk is processed. This keeps GPU memory consumption proportional to C rather than to the full table size S, avoiding the out-of-memory error that occurs when the original code attempts to move all $S = 16\,777\,216$ rows to the GPU at once. Gradient computation is disabled using `torch.inference_mode()` rather than the original `torch.no_grad()`, which further reduces memory overhead during evaluation.

5.3.2. Adaptation 2: Direct LUT Addressing

Original behaviour

During LUT-mode inference, to propagate an input combination through all the perceptrons to the MLP, the original lookup function computed the output for a given input by scanning every entry in the truth table, comparing the input against each stored entry until a match was found. For a table with $S = 16\,777\,216$ entries, this full scan is repeated for every perceptron on every inference step, making LUT-mode inference completely impractical at 12-bit precision.

Adaptation

Because the truth table is filled in a predictable order, the position of any entry can be calculated directly from the input values without searching. The entries are placed in a lexicographic order, meaning placed in the same fixed sequence every time.

The quantised inputs produced during inference are signed integers, so the most negative value does not naturally correspond to position zero. A fixed offset is added to each input to shift the range so that

the most negative value maps to zero and all values become non-negative. The two offset inputs are then combined into a single table position, exactly like reading a coordinate on a grid: one input selects the column and the other selects the row, and the flat position is row $\cdot\, W +$ column, where W is the number of possible values per input. With 12-bit inputs, $W = 2^{12} = 4096$, giving $4096^2 = 16\,777\,216$ unique positions, one for every possible input pair. Expressed in terms of bit operations, this is:

$$\text{index} = \sum_{k=0}^{F-1} \text{input}_k \ll (k \cdot B), \tag{5.13}$$

where $\ll$ denotes a left bit-shift (equivalent to multiplying by a power of two). The truth-table output is then retrieved with a single direct read, reducing the lookup cost from a full scan of all S entries to a fixed number of operations that does not grow with table size.

5.3.3. Adaptation 3: Parallelised Verilog Generation

Original behaviour

Generating a Verilog file requires writing one line of text, a case statement row, for each entry in the truth table that shows what output value corresponds to which input combination value. The original Python code wrote these lines one at a time in a Python loop. Python, unfortunately, evaluates each instruction individually at runtime, for simple loops by default. At $S = 16\,777\,216$ entries per perceptron, across all perceptrons in all layers, the sheer number of iterations made this a dominant bottleneck of the conversion pipeline, but also a classic textbook problem to solve.

Adaptation

The loop was rewritten using Numba, a library that compiles NumPy code to optimised machine code before it runs, achieving speeds comparable to those of languages like C. Inside the compiled loop, entries are encoded as integers using arithmetic operations rather than text characters as in Python, since arithmetic is much faster. Text formatting is used only after the loop completes, when all entries are converted to strings in a single batch operation. Additionally, the perceptrons within each layer are processed simultaneously across all available CPU cores, enabling the conversion of the maximum number of perceptrons in parallel rather than sequentially on a single core.

Together, these three changes reduce the conversion time from a multi-week runtime to a tractable duration of a few hours on a standard workstation.

5.4. ReducedLUT: LUT Compression

Although the NeuraLUT conversion enables FPGA implementation of LUT-based neuron models, the resulting truth tables remain large because the tool maps them as monolithic case statements. While direct LUT implementation in embedded logic blocks on the FPGA provides fast inference access, the required hardware resources for these blocks saturate the capacity of the target FPGA devices in this work.

ReducedLUT [18] addresses this problem by exploiting the structure present inside the generated truth tables. As discussed in Section 2.3.3, instead of storing every input combination separately, the algorithm searches for repeated patterns, specifically equivalent subtables, removes redundant information and generates a resulting compressed hierarchical representation that can be synthesised into a significantly smaller number of logic blocks.

The original ReducedLUT implementation, however, was designed for general LUT compression and thus perfectly compatible with the low-resolution LUTs from NeuraLUT. Applying it directly to the high-resolution Hodgkin-Huxley LUTs exposes several computational and memory limitations. The original algorithm relies on expensive subtable-similarities searches and stores large intermediate structures. For the 2^{24}-entry LUTs required in this work, these operations become impractical. Therefore, four targeted adaptations were implemented to make ReducedLUT suitable for large-resolution LookUp Tables.

5.4.1. Adaptation 1: Compact Storage of 'care' values

Original behaviour

The original code tracked which table entries were observed using two separate structures: a hash map recording how often each observed input was seen, and a set explicitly listing all unobserved (don't-care) indices. These two structures together hold two complementary halves of the same binary information, namely whether each entry was seen or not, but each carries substantial internal overhead per element for hashing, pointers, etc. At 16.7 million possible inputs, storing this information twice in high-overhead containers causes the memory consumption to far exceed the LUT data itself.

Adaptation

Both structures are replaced by a single flat byte array with one entry per table position: 1 if the input was observed, 0 if not. This stores exactly the same information as the two original structures combined, at one byte per entry with no overhead, capping memory usage at 16.7 MB regardless of how many entries are cares or don't-cares. This representation removes dynamic allocation overhead and redundant memory consumption, making it suitable for the large truth tables generated by NeuraLUT.

5.4.2. Adaptation 2: Subtable Pattern Matching

Original behaviour

The main compression step of ReducedLUT divides the truth table into smaller subtables. Similar subtables can then share a single stored representation. The original implementation searched for these similarities by constructing a subtable similarity matrix. Each subtable was compared with every other subtable, accounting additionally for possible bit shifts, as shown in Figure 2.19.

Although this approach can achieve high compression, its computational and memory costs grow quadratically with the number of subtables. For a LUT containing N subtables, the similarity matrix requires N^2 comparison entries. With $N = 2^{16}$ subtables, this results in 2^{32} entries, making the approach impractical for the high-resolution LUTs used in this work.

Adaptation

The pairwise similarity search was replaced with a dictionary-based pattern-matching approach. This modification applies the principle of indexed pattern reuse, in which previously encountered structures are identified via a canonical representation and retrieved by direct lookup rather than repeated comparison [64]. Specifically in our context, the $N \cdot N$ similarity matrix is replaced by a dictionary that maps each normalised subtable pattern to its position in the list of unique patterns stored so far. Before matching, each subtable is normalised by removing its minimum value (bias) and applying possible bit shifts. Bias subtraction and shifting are performed only on care entries, while don't-care positions are replaced with zeros. This gives don't-care positions a fixed representation, allowing subtables with identical care values but different unused positions to share the same stored pattern.

The resulting pattern is then checked against previously stored patterns:

- If the pattern **already exists**, the subtable stores only an index pointing to the existing pattern.
- If the pattern is **new**, it is added as a new entry to the unique-pattern table and the new index is stored.

This changes the matching process from a quadratic number of pairwise subtable comparisons into incremental dictionary-based matching, where each new subtable is compared only against the set of unique patterns.

The original ReducedLUT method additionally used don't-care values as wildcards, allowing subtables with different values at unused positions to still be considered as equivalent. This behaviour is not directly supported by standard dictionary matching, since dictionary comparison requires the complete stored pattern to match exactly. For example, a care value of 6 in one subtable, and a don't care value represented as 0 in another subtable are treated as different subtables, even though the original wildcard-based method would consider them equivalent. Therefore, an optional extended matching mode was added. In this mode, only care positions are compared, while don't-care positions are ignored during matching. This increases compression opportunities but requires checking individual care entries, thereby increasing the computational cost. This process is concisely showcased in the subtable

pattern matching stage in Figure 5.5. The full process of the adapted ReducedLUT, including low-bits separation, subtable pattern matching, multilevel bias table decomposition, and the reconstruction of the original value, is shown in Figures A.1 to A.3 in Appendix Chapter A.

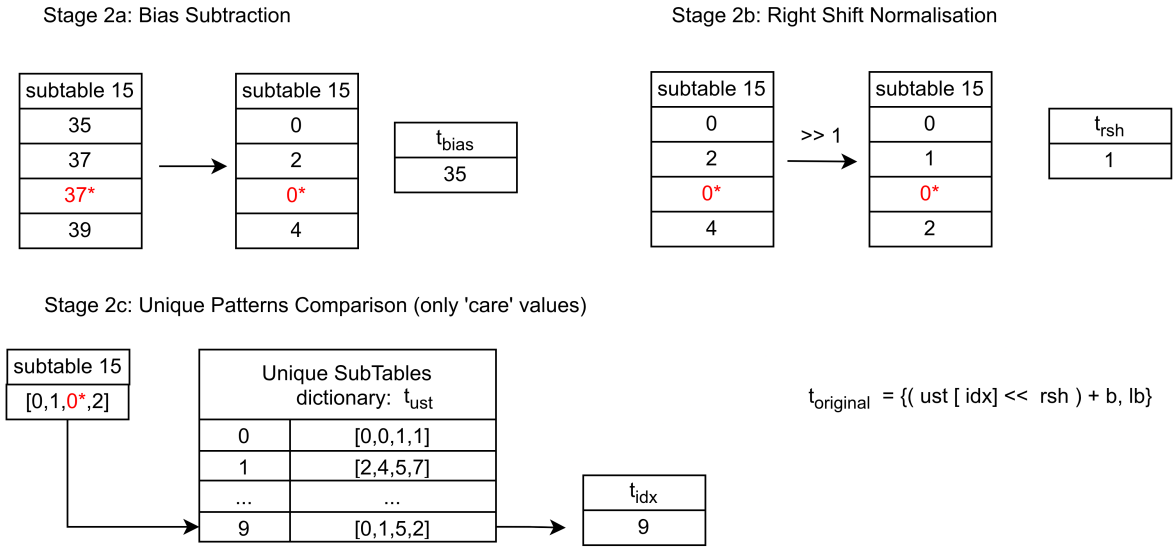


Figure 5.5: Sub-process visualisation of the adapted ReducedLUT algorithm showcasing the subtable pattern matching (stage 2). Full process can be found in Appendix Chapter A.

Finally, the preprocessing stage of each subtable was parallelised across available CPU cores. Since each subtable can be normalised independently before the pattern-matching stage, the expensive preparation step can be distributed across multiple threads, reducing the overall compression runtime.

5.4.3. Adaptation 3: Noise Safety Margin

The HH state variables follow specific trajectories through their value space during simulation. As shown in Figure 5.6, when simulating the gate channels in relation to the voltage state, it can be observed that, despite a wide range of different stimulus scenarios, the variables visit only a limited region of all possible input combinations (V-n, V-m, and V-h combinations). These loops of possible outputs are called the limit cycles of the state variables.
This makes the don't-care optimisation particularly effective here, since the occupied region is a small fraction of the full 2^{24} input space, and a large remainder is marked as redundant.

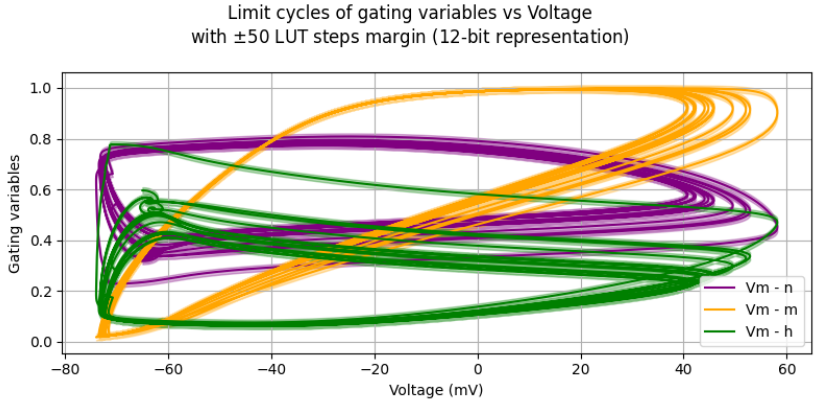


Figure 5.6: Visualisation of the value-space of the gate channel variables, showing the limited region of the 2^{24} input space that is actually visited, in combination with a margin of 50 quantisation steps around it, which was used as the *care* mask in the ReducedLUT algorithm.

Original behaviour

The original algorithm, however, shows no tolerance for unused inputs. Any input present in the log, even one differing from an observed input by a single quantisation step, is immediately marked as a don't care. During hardware inference, small numerical differences due to quantisation, rounding, or accumulated calculation errors can push the inputs just outside of the observed region, causing the LUT to return incorrect values. Moreover, even though the limit cycles stay close to the same trajectory, every different stimulus shifts the cycle just a little bit out of the original position in the graph. With the original algorithm, confidently marking all relevant input combinations as 'care' values requires simulating all possible stimulus values to capture all possible limit cycles, which is impractical.

Adaptation

To ensure that all possible limit cycles are covered even after compression while eliminating redundant inactive regions, a configurable safety margin was added around all care entries. This implementation performs this expansion along both input dimensions of the two-input LUTs. For every observed entry, all entries within a set number of steps along each axis in both directions are additionally marked as cares. This creates a buffer zone around the observed region, ensuring that small deviations are still handled correctly by the compressed LUT.

In this work, all LUTs were compressed using a margin of 50 quantisation levels per input dimension in the discrete input grid used for LUT addressing. The margin is applied to 50 levels above the true quantisation step and 50 levels underneath. This buffer ensures that small deviations caused by quantisation effects or numerical accumulation do not move inputs outside the valid region learned during compression. While this choice reduces the maximum achievable compression ratio, it improves robustness of the Hodgkin–Huxley dynamics under discretisation effects and prevents incorrect LUT lookups near the boundary of the observed state space.

5.4.4. Adaptation 4: Pre-shifted Final Data for Synthesis

Original behaviour

During compression, ReducedLUT stores the compressed values in binary. However, this does not always produce the most efficient FPGA implementation. Output values often contain unused least-significant bits after compression, especially when values are derived from biased or shifted representations. The synthesis tool receives a signal that is always zero at this bit position, but must still treat it as a live wire, wasting routing resources.

Adaptation

During compression, the algorithm determines the maximum number of redundant trailing zero bits shared across all those care entries, and removes these bits from the normalised subtable values by applying a right shift:

$$y_{compressed} = y_{original} \gg n, \tag{5.14}$$

where n represents the number of discarded least-significant bits.

The removed shift amount is stored together with the compressed subtable representation in a separate shift table. During RTL generation, this metadata is used together with the stored pattern index and bias value to reconstruct the original numerical alignment.

By removing unnecessary lower bits before storage, the compressed patterns require fewer bits. This reduces the size of the generated LUT representation and allows the FPGA synthesis tool to implement the resulting logic more efficiently.

Together, these four adaptations transform ReducedLUT from a general-purpose small LUT compressor into a scalable compression pipeline suitable for high-resolution (HH model) lookup tables. The resulting representation reduces memory requirements while maintaining the deterministic lookup behaviour required for real-time FPGA Hodgkin-Huxley simulation.

5.4.5. ReducedLUT Parallelisation

The Python wrapper around the ReducedLUT executable was also adapted. The original wrapper processed neurons sequentially, running the compression tool on one neuron at a time and waiting for each to finish before starting the next. Since each neuron's truth table is entirely independent, the adapted wrapper submits all neurons as concurrent jobs distributed across all available CPU cores. This reduces the total compression time by a factor of the number of available cores.

5.4.6. Discussion on Adaptation Performance

The adaptations introduced in this work for both NeuraLUT and ReducedLUT are motivated by a single primary objective: enabling them to operate on high-resolution (12-bit per variable, 2^{24}-entry) Hodgkin–Huxley lookup tables within practical memory and runtime constraints on available hardware. The original implementations of both toolchains were developed for significantly lower-resolution regimes, where exhaustive truth-table generation and pairwise subtable comparison remain computationally tractable. When applied directly to the resolution required in this work, both memory consumption and runtime become prohibitive, preventing successful execution. The modifications presented in this thesis, therefore, focus on reducing memory overhead, avoiding quadratic or full-space operations, and introducing structured representations that make large-scale LUT generation and compression feasible.

The goal of these adaptations is not to propose improved or competing versions of the original algorithms, nor to provide a comparative evaluation against their low-resolution performance. Instead, the objective is to preserve the functional behaviour of the original methods while enabling execution at a substantially larger scale. As a result, no systematic benchmarking against the original implementations is performed, and no claims are made regarding relative performance improvements across equivalent low-resolution settings. The effectiveness of the adaptations is validated indirectly through successful generation, compression, and synthesis of the required 12-bit Hodgkin–Huxley LUTs within practical runtime limits (from infeasible multi-week execution estimates to sub-hour processing times in the final pipeline), rather than through direct comparison studies.

5.5. Verilog: Neuron Core

5.5.1. Pipeline

The neuron-core simulation combines all Neural Network-derived LUT networks for the Hodgkin–Huxley subcomponents into a unified hardware system. This structure follows the system architecture defined in Chapter 3, where each functional block is implemented as a sequence of LUT-based network stages. Each LUT layer is already implemented with explicit register boundaries introduced during the NeuraLUT conversion, so the resulting computation naturally forms a staged dataflow that can be mapped onto a pipelined architecture.

While LUT evaluations and DSP-based arithmetic operations have different inherent latencies, the pipeline is explicitly balanced to avoid long combinational paths in the multiplier chain. In particular, only one DSP multiplication is scheduled per pipeline stage, ensuring that multiplication latency does not dominate the clock period or require additional stage splitting beyond the LUT structure.

The overall pipeline depth is then determined by the longest aligned dependency chain across all submodules, including gate-channel evaluation, ionic-current computation, and voltage update. On top of the neural-network LUT layers, which are integrated as stages, additional stages are required for Euler forward integration, in which the scaled state updates are accumulated and written back into the state registers. The pipeline is therefore constrained by both LUT layer structure and DSP scheduling, rather than by either component alone.

In each pipeline stage, independent hardware resources are available, allowing multiple neuron states to be processed concurrently across different stages. The pipeline for Design 4, including all optimisations from Chapter 3, is shown in Figure 5.7. It spans 6 stages (M0 to M5) to complete a state update.

Design 3 follows the same pipeline depth because the ionic-current computation is also implemented as a multi-layer LUT network with a stage structure equivalent to the gate-channel networks, resulting in aligned timing across submodules. In contrast, Designs 1 and 2 were not implemented in hardware due

to substantially larger and wider LUTs for the voltage update, which would require a different balance between stage depth and memory footprint.

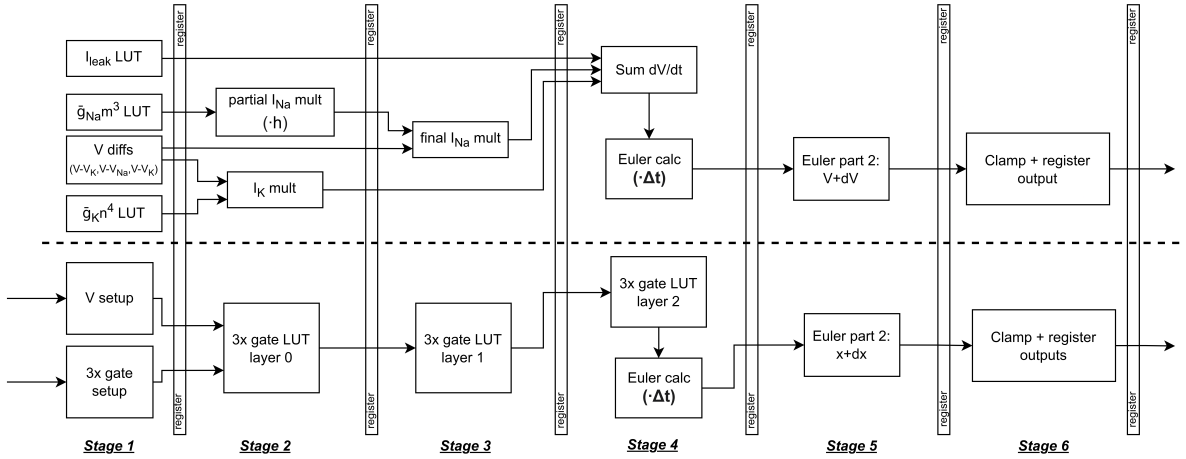


Figure 5.7: Visualisation of the 6 pipeline stages of the neuron core with Design 4 [HYBRID] implemented in hardware (Verilog), where the blocks for the gate-channels represent a threefold amount of blocks for each of the gate channels n,m, and h

6

Neuronal Network

The previous chapters focused on the design, optimisation and implementation of the Hodgkin-Huxley neuron core, performing state-update calculations for individual discrete time steps of a single neuron. However, scaling from a single neuron core towards a neuronal network introduces additional challenges beyond the computational complexity of HH dynamics. In particular, the storage, scheduling, and communication between all neurons and synaptic paths must be efficiently managed for large-scale simulation.

This chapter presents the integration of the developed neuron core into a real-time neuronal network. The network architecture is based on time-multiplexing a single neuron core across multiple simulated neurons while maintaining the simulation timestep of 0.01ms. Dedicated memory structures are introduced to store neuron states, synaptic connections, and per-neuron stimulus accumulation. Together with a spike-routing mechanism, these components form the infrastructure required for network-level simulation.

The use of a single time-multiplexed core serves as a validation platform for the scheduling logic, memory subsystem, and spike-routing mechanism. It provides a controlled environment to verify correct system behaviour before considering replication across multiple parallel cores, a capability supported by the implementation and consistent with the resource usage reported in later chapters.

The scope of this work is limited to a proof-of-concept implementation. The primary objective is to demonstrate the feasibility of the proposed neuron core within a network setting and to identify system-level constraints. Full optimisation and large-scale deployment of a multi-core neuromorphic system are left for future work.

6.1. Top-Level Architecture

The network adopts a Brunel-style connectivity regime [28], in which excitatory and inhibitory neurons are connected through sparse recurrent synaptic projections. In this work, sparsity refers to a fixed probability of connection between any pair of neurons, resulting in a randomly connected graph with low connection density. The selected configuration consists of 1.250 neurons, divided into 80% excitatory and 20% inhibitory populations, with a synaptic connectivity probability of 10%. This configuration provides a representative sparse network structure that stresses spike-routing patterns while remaining feasible for a proof-of-concept implementation. It enables evaluation of the scalability of the proposed neuron core under realistic communication load, independently of the underlying neuron dynamics defined by the Hodgkin–Huxley model.

The system combines two execution regimes. Neuron-state updates are executed deterministically at each simulation timestep via a time-multiplexed neuron-core pipeline, yielding a fixed, predictable computational workload. In contrast, synaptic communication is event-driven: only neurons that generate a spike trigger routing activity, and the number of routing operations varies with network activity. This separation allows neuron updates to maintain a constant execution rate, while spike propagation

introduces a workload that depends on network dynamics.

The complete neuronal network consists of four main components: neuron-state storage, synapse storage, the neuron-core wrapper, and the spike router. Since a single neuron core is reused for multiple neurons, the system operates via a single pipelined neuron core that runs in a time-multiplexed fashion. During each timestep, neuron states are loaded from memory into the neuron core pipeline each clock cycle, then written back after processing is complete. In parallel, synaptic events generated by spikes are accumulated for each neuron for the next update cycle. Figure 6.1 shows a high-level overview of all the components and simplified connections between them.

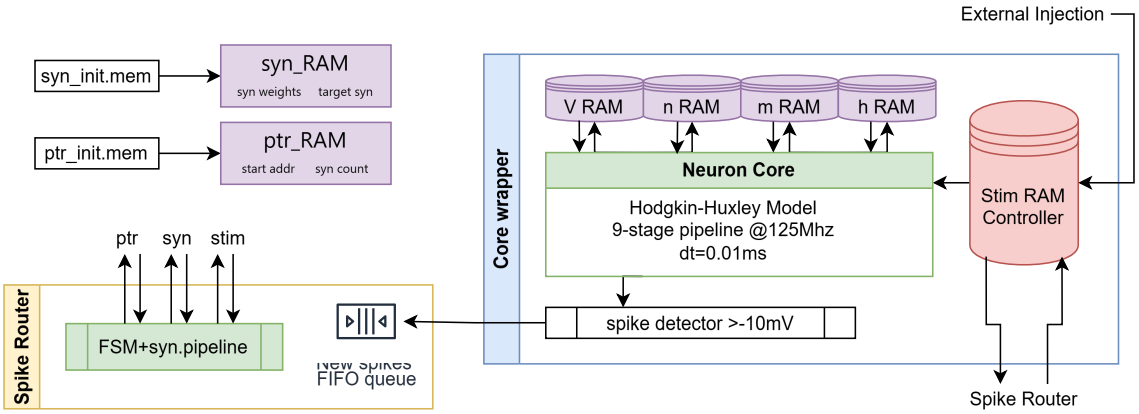


Figure 6.1: Schematic high-level representation of the top-level architecture in hardware, featuring the four main components: Neuron-Core Wrapper (blue), Spike Router (yellow), 6 different RAMs for neuron-state and synapse storage (purple), and a unified Stim RAM Controller (red). Additionally, two pipelines exist in the system (green): one within the Spike Router, combined with an FSM and a preceding FIFO queue, and one within the Neuron-Core Wrapper, which exploits the Neuron Core to implement a 9-stage pipelined process operating at 125 MHz and simulating an update per time-step $dt = 0.01ms$. Moreover, the Neuron-Core Wrapper includes a spike detector to trigger the Spike Router to send synaptic weight updates to connected neurons.

6.2. Neuron-Core Wrapper

The neuron-core wrapper provides the scheduling and memory interface around the pipelined neuron core, enabling integration into a multi-neuron time-multiplexed system. The neuron core implements a fixed 6-stage computation pipeline, producing an updated set of neuron-state variables every clock cycle once the pipeline is filled.

The wrapper sustains this throughput by managing BRAM-based state storage, coordinating data movement into and out of the core, and aligning memory transfers with the core's pipeline stages. It also contains an observer module (spike detector) that identifies threshold crossings of the membrane potential at $-10\,\mathrm{mV}$, indicating the initiation of an action potential. Detected events are forwarded to the spike-routing subsystem for synaptic communication between neurons. Spike events are extracted in discrete time by comparing successive samples of the membrane potential. Although the Hodgkin–Huxley model produces continuous-time dynamics in the biological formulation, the hardware implementation evaluates these dynamics at fixed timesteps. This requires an explicit event-detection mechanism to translate continuous state evolution into discrete communication events.

The wrapper integrates all memory structures required for state evolution, including four BRAMs for state variables and two BRAMs for stimulus accumulation. While the neuron core has previously been shown in Section 5.5 to require only 6 cycles for a full computation, the wrapper extends the effective pipeline to 9 stages due to BRAM read latency, input staging for alignment with the core pipeline, and write-back latency for updated states. These additional stages ensure continuous utilisation of the core by providing new inputs every cycle without stalling.

Neuron- and Stimulus-State Storage

Efficient memory organisation is essential when scaling the neuron core towards larger networks. The neuron-core wrapper contains the neuron state memory and stimulus memories required for neuron

updates, while the synaptic connectivity is stored separately as a dedicated global memory structure used exclusively by the spike-routing subsystem. This separation decouples local neuron state updates from inter-neuron communication, allowing the neuron core and its wrapper to operate only on per-neuron state variables, while the spike router accesses the connectivity data to distribute synaptic events to the correct target neurons when a spike occurs.

The neuron state memory stores the dynamic variables n, m, h, and V of the HH model, while separate stimulus memories store incoming synaptic and externally injected currents. To separate synaptic updates from neuron processing, a ping-pong BRAM structure (double buffer) is used as stimulus memory. During each simulation timestep, one stimulus BRAM acts as the active bank and is accessed sequentially by the neuron-core pipeline. After the stimulus value of a neuron has been read, the corresponding location is cleared so that a new stimulus state can be accumulated there in future timesteps.

Simultaneously, the second stimulus BRAM acts as an accumulation bank and is connected to the spike router. Whenever a neuron spikes, the router retrieves all outgoing synaptic connections and accumulates their corresponding weights into this inactive bank. At the end of the simulation timestep, both stimulus BRAMs swap roles. The inactive bank with accumulated synaptic contributions becomes the active bank for the next neuron-core update cycle.

This ping-pong architecture eliminates memory-access conflicts between neuron updates and synaptic-event propagation while ensuring that synaptic events influence neuron dynamics no earlier than the following simulation timestep. Because all synaptic updates are written into a separate buffer, spike propagation can continue independently of the neuron-core pipeline progression.

In cases where spike generation rates exceed the available accumulation bandwidth within a single timestep, newly generated events may be deferred into the subsequent accumulation cycle. This does not represent yet a variable spike-delay mechanism, but rather a temporary backlog in synaptic event processing under saturation conditions. Consequently, accumulated events are not lost but may be incorporated one timestep later once processing capacity becomes available.

While one BRAM is accessed by the neuron core for the state update and reset after reading, the second bank connects to the spike router and accumulates incoming synaptic events for the neuron at the next timestep. This ensures that spikes influence connected neurons in the next simulation timestep, matching the discrete-time behaviour of the simulated network. The full functionality of both BRAMs and the double-buffer process is visualised in Figure 6.2. In addition, an additional mechanism is integrated to handle synaptic exponential decay, as shown in the figure as the *Decay Stim RAM*. This mechanism is elaborated in Section 6.2.2.

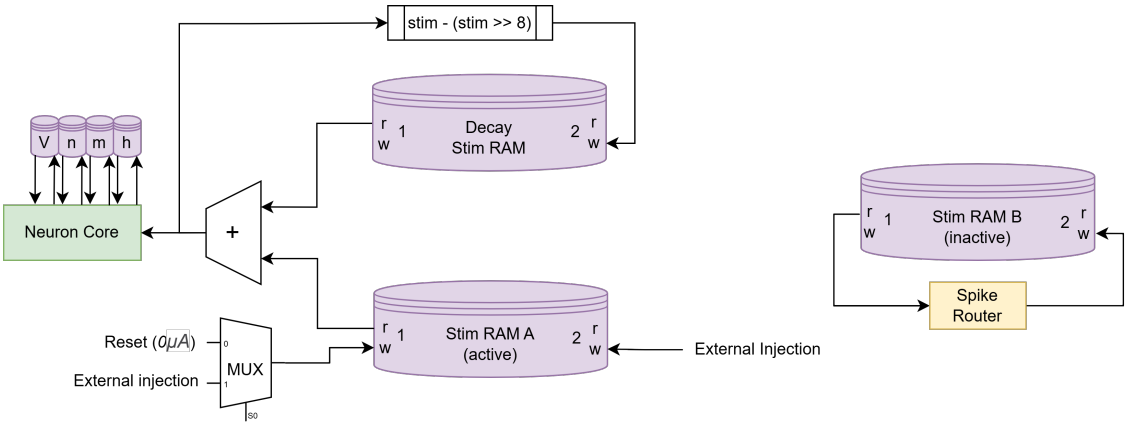


Figure 6.2: Functionality and dataflow overview of the *Stim RAM Controller*, including the two alternating stimulus RAMs (active/inactive) and the stimulus decay RAM. At each simulation step, the input to the neuron core is the sum of the exponentially decaying stimulus from the previous timestep and the accumulated synaptic contributions stored in the currently active RAM. The active and inactive RAMs are swapped every timestep, with synaptic updates written into the inactive buffer via the spike router and external inputs to the active buffer. After each update cycle, the accumulated synaptic values are reset, and the roles of the two RAMs are exchanged.

6.2.1. State-Accumulator Resolution

The resolution of the state accumulators affects both numerical accuracy and FPGA memory usage. Although the LUT interfaces operate with 12-bit fixed-point values, the internal Euler accumulators require higher precision to preserve the accumulated state evolution introduced by derivative-based prediction. This requirement for the state-accumulator is discussed in Section 4.3.1.

However, for a target network size of 1.250 neurons, storing 24-bit state vectors results in inefficient BRAM utilisation. Since a 36Kb-BRAM can store up to 2048 entries at $\leq$18-bit width but only 1024 entries at wider configurations, a 24-bit accumulator array exceeds the native depth limit and requires an additional 18Kb-BRAM block [65].

Reducing the accumulator width to 18 bits allows the complete state vector to fit within a single 36Kb-BRAM, significantly reducing memory overhead. This maintains six additional bits of precision compared to the 12-bit neuron states, still providing increased numerical margin for the forward Euler propagation while avoiding the additional BRAM cost of the 24-bit representation.

6.2.2. Stimulus Exponential Decay

To preserve synaptic effects across multiple hardware cycles, incoming stimuli are accumulated using a discrete-time, exponential-decay model [66]:

$$s[n+1] = s[n]\alpha + x[n] \tag{6.1}$$

where $s[n]$ represents the accumulated stimulus state, $x[n]$ is the incoming spike-driven input, and α determines the decay rate.

Implementing the decay using a fixed-point multiplication would increase hardware cost. Therefore, the exponential decay is approximated using a shift-based leaky integrator:

$$s[n+1] = s[n] - (s[n] \gg k) + x[n] \tag{6.2}$$

which corresponds to a decay factor $\alpha = 1 - 2^{-k}$. For the implemented design, $k = 8$, resulting in

$$\alpha = 1 - \frac{1}{256} = 0.99609$$

This implementation replaces multiplication with a simple shift-and-subtract operation, significantly reducing hardware complexity. With a timestep of ($\Delta t = 0.01, \mathrm{ms}$), the equivalent continuous-time decay constant can be obtained from:

$$\alpha = e^{-\Delta t/\tau}, \tag{6.3}$$

yielding

$$\tau = \frac{-\Delta t}{\ln(\alpha)} = \frac{-0.01, \mathrm{ms}}{\ln(0.99609375)} \approx 2.55\mathrm{ms} \tag{6.4}$$

The decay does not introduce a hard cutoff; instead, it defines an effective memory horizon over which past synaptic contributions remain relevant. For example, if a 1% significance threshold is used, the influence of a single input decays to a negligible level ($< 1\%$) after approximately $n \approx 1175$ simulation steps, corresponding to about $11.75\,\mathrm{ms}$ at the chosen timestep. This provides a finite but smoothly fading memory window for synaptic integration while avoiding abrupt loss of accumulated stimulus. This type of hardware-oriented approximation is consistent with previous work investigating low-cost approximations of exponential decay in neuromorphic hardware [67].

Regarding its placement in the neuron-core wrapper architecture, the exponential decay is implemented in a dedicated stimulus-decay BRAM that stores the synaptic state for each neuron. During neuron

evaluation, the decayed stimulus state is added to the newly accumulated synaptic input from the stimulus accumulation memories. The resulting stimulus value is supplied to the neuron core and subsequently written back to the decay memory after applying the shift-based decay operation. This operation is visualised in Figure 6.2.

6.3. Synaptic Connectivity

When a neuron spikes, the spike router retrieves all connected target neurons and applies the corresponding synaptic weight. Depending on whether the spiked neuron is excitatory or inhibitory, it applies either small positive stimulus increments or larger negative contributions to compensate for the smaller inhibitory population. These weight magnitudes were selected to achieve stable network dynamics, and were determined empirically through iterative simulation runs in which parameters were adjusted until the network exhibited sustained spiking activity without firing rates growing without bound or dropping to zero.

Instead of storing a fully connected synaptic matrix with entries containing source neuron, target neuron, and the corresponding synaptic weight, the network uses a sparse-connectivity representation inspired by compressed sparse row (CSR) storage [68]. As shown in Figure 6.3, connectivity is divided over two BRAM structures: the pointer RAM and the synapse RAM.

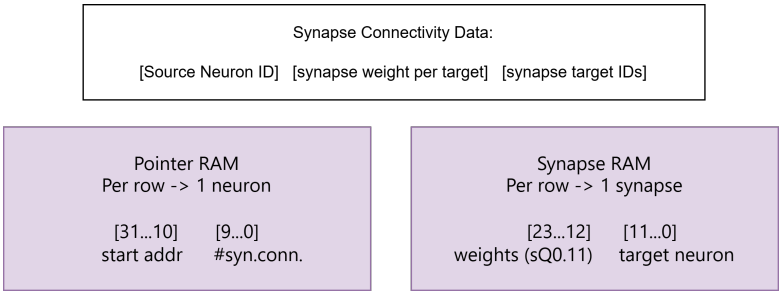


Figure 6.3: Schematic of the split compact representation of the synaptic connections between neurons using 2 BRAMs

The pointer RAM stores one entry per neuron. Each entry contains the starting address of the neuron's connections in the synapse RAM and the number of outgoing synapses. The 22-bit address field allows approximately (4.2×10^6) synapse entries to be addressed, while the 10-bit connection counter supports up to 1024 connections per neuron. For the current network size of 1.250 neurons with 10% connectivity, only approximately (1.6×10^5) connections are required, leaving sufficient capacity for future scaling. The synapse RAM stores one directed connection per entry, consisting of the target neuron ID and its associated synaptic weight. In the implemented network, at most one weighted connection is stored between any pair of neurons. This follows the connectivity model adopted for the proof-of-concept network, in which each connection is represented by a single effective synaptic weight rather than multiple parallel synaptic contacts. Supporting multiple parallel synapses between the same neuron pair would simply require storing multiple entries with the same target neuron ID or extending the connection representation, but this was outside the scope of the present implementation.

Both values use 12-bit representations. The target ID supports up to 4096 neurons, so larger networks would require an increased address width. The synaptic weight uses a fixed-point representation, where the most significant bit indicates the sign and the remaining 11 bits represent the fractional magnitude, providing a resolution of ($\frac{1}{2^{11}} \approx 0.00049$). This precision is higher than required for the current stimulus range, meaning that the representation could be reduced if BRAM utilisation becomes a limiting factor. To introduce variability among synaptic connections and avoid overly uniform network behaviour, each stored weight is generated by applying a uniformly distributed ±20% variation around the predefined base weight.

To introduce variability among synaptic connections and avoid overly uniform network behaviour, each connection weight is generated by applying a uniformly distributed $\pm 20\%$ variation around a nominal excitatory or inhibitory weight. Since the inhibitory population comprises only 20% of the neurons, inhibitory synapses are assigned a nominal magnitude approximately 5 times that of excitatory synapses

to maintain the desired excitation-inhibition balance, consistent with the commonly adopted approach for balanced spiking networks. The nominal weights were determined empirically through iterative simulation runs until sustained network activity was achieved, after which random variation was applied to each connection. The resulting weight values are reported and evaluated in the network results chapter.

6.4. Spike Router

After the neuron core generates a spike, the spike router propagates the corresponding synaptic updates to all connected target neurons. Each spike ID is first used to access a pointer memory containing the start address and number of outgoing synapses. The router then sequentially reads the corresponding synapse entries, which contain the target neuron ID and synaptic weight, and accumulates the weight into the target neuron's stimulus memory. A simplified overview of how this is done is shown in Figure 6.4.

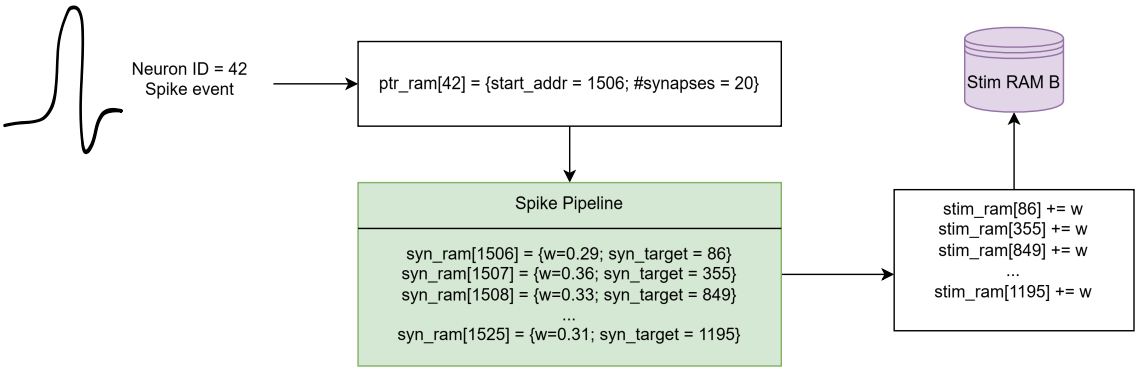


Figure 6.4: Simplified visualisation of the event-driven synaptic update mechanism after a neuron has spiked. In this example, a triggered spike event from neuron 42 is propagated to the Spike Router, where the pointer RAM first provides information stored for this 'source' neuron, namely the start address in the synapse RAM and the number of neurons it is connected to. Then, through a pipeline (green), the Spike Router iterates over the connected neurons and looks up the synaptic weight for this connection. Afterwards, the weight is added to the current accumulated stimulus value from the stim RAM and written back to it.

Since a single spike can trigger many synaptic updates, the router is implemented using a pipelined streaming architecture. Incoming spikes are stored in a FIFO buffer, allowing the neuron-core pipeline to continue operating while previously generated spikes propagate. The router, therefore, decouples the deterministic neuron-update pipeline from the variable workload imposed by spike propagation.

The propagation process consists of two main stages: an FSM that controls spike fetching and synapse traversal, followed by an accumulation pipeline that performs the stimulus update. After pipeline filling, synaptic updates can be issued continuously, enabling one target update per clock cycle. The overall architecture is shown in Figure 6.5.

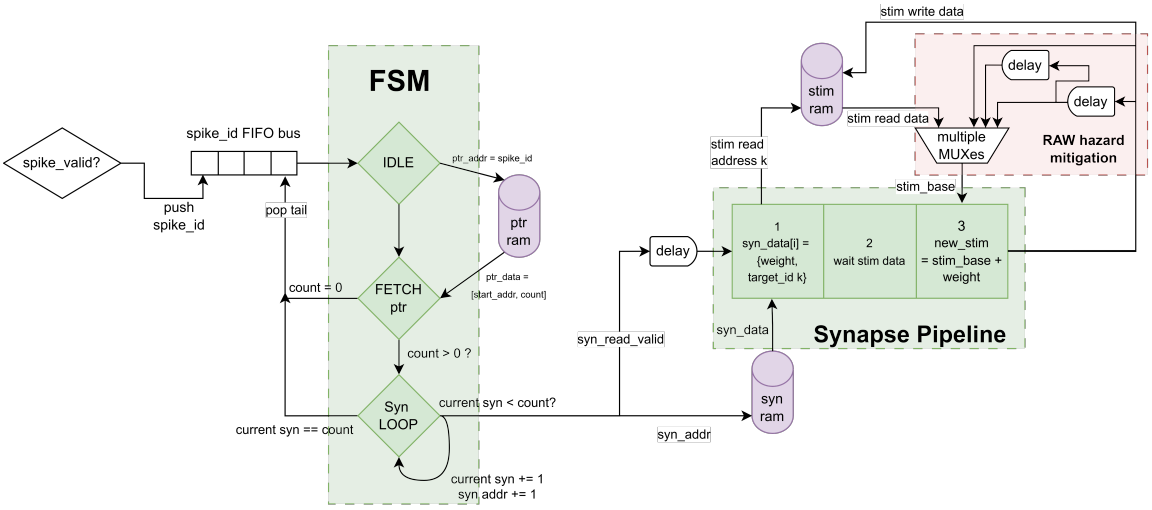


Figure 6.5: Architectural block diagram of the Spike Router, as part of the neuronal network architecture. The flow illustrates the integration of a FIFO buffer for spike queueing, a finite state machine (FSM) for 'source' neuron information acquisition, and a multi-stage synapse pipeline with a hazard mitigation loop.

Spike FSM

The Spike FSM manages the lookup of the spiking neuron's connectivity information and is visualised in Figure 6.6.

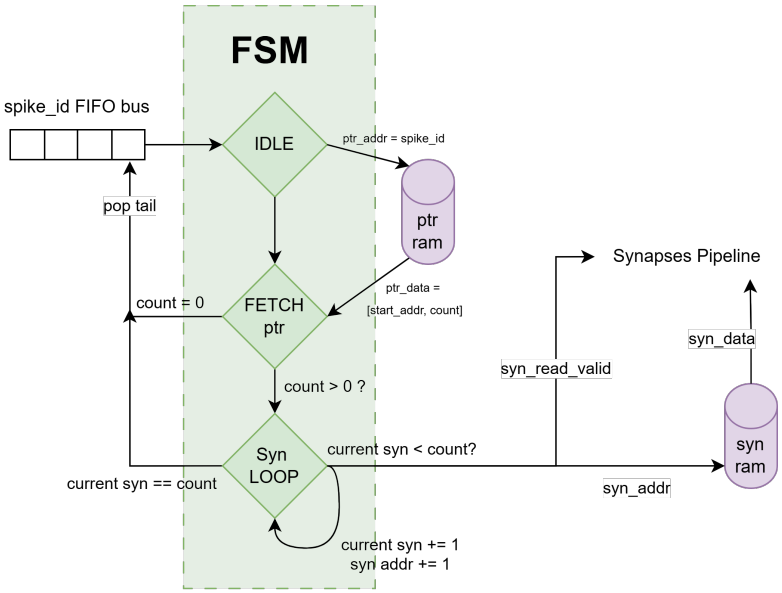


Figure 6.6: Architectural block diagram of the spike FSM, as part of the neuronal network architecture. The flow illustrates the integration of a FIFO buffer for spike queueing, which passes 'source' neurons to a finite-state machine (FSM). The FSM then looks up information for this neuron in the pointer RAM. In the 'FETCH ptr' state, the FSM receives information from the connected neurons. In the 'Syn LOOP', the FSM loops over all connected neurons and passes them to the synapse pipeline to update all stimulus values.

When a spike is available in the FIFO, the corresponding pointer entry is fetched from the pointer RAM to determine its starting memory address in the synapse RAM and the number of outgoing synapses. The FSM then iterates over the synapse memory until all target neurons have been processed, after which the spike is removed from the FIFO. For each outgoing synapse, the FSM sends a validation signal to the FSM pipeline and the address to the synapse RAM; thereafter, the data from this RAM

also goes to the pipeline, where it's processed. However, since it's a pipeline, the FSM sends a new synapse address and signal each cycle until the count received from the pointer RAM is reached.

The Spike FSM manages the lookup of the spiking neuron's connectivity information. When a spike is available in the FIFO, the corresponding pointer entry is fetched from the pointer RAM to determine its starting address in the synapse RAM and the number of outgoing synapses. The FSM then iterates through the synapse memory until all target neurons have been processed, after which the spike is removed from the FIFO.

For each synapse, the FSM issues the synapse memory address, along with a validation signal, to the processing pipeline. After the initial overhead, new synapse requests are sent at every clock cycle until the spike's complete fanout has been processed.

Synapse Pipeline

The synapse pipeline converts each synapse entry into a stimulus memory update. A synapse entry contains the target neuron index and the associated fixed-point weight. The target stimulus value is read from the stimulus RAM (inactive bank), the synaptic weight contribution is added, and the updated value is written back.

Because consecutive synaptic updates may access the same target neuron, the pipeline may encounter a read-after-write (RAW) data hazard. This occurs when a new update reads a target neuron's stimulus value before a previous update to the same neuron has been written back. To resolve this, forwarding logic is implemented: if the target neuron is already being updated in an earlier pipeline stage, the most recent intermediate value is forwarded instead of the original stimulus memory value. This ensures that multiple synaptic inputs targeting the same neuron are accumulated correctly while maintaining a fully pipelined update flow.

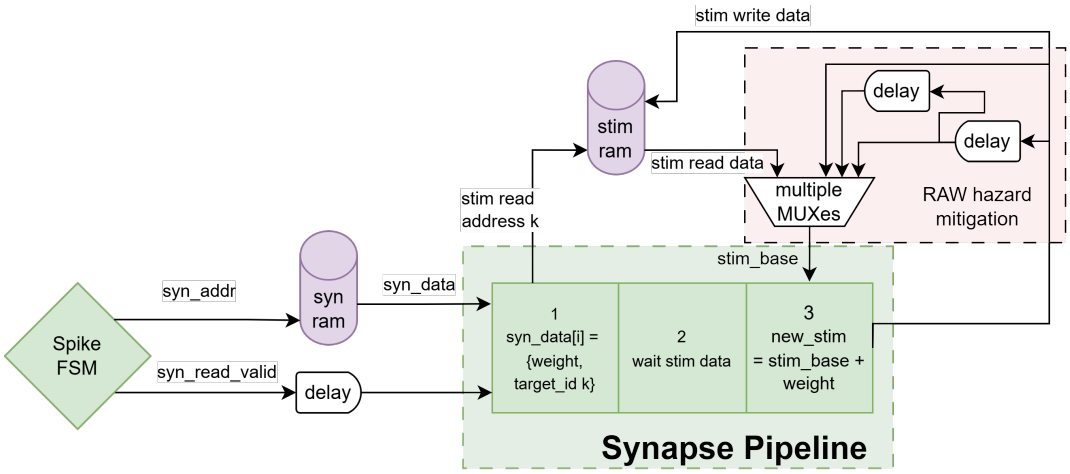


Figure 6.7: Architectural block diagram of the spike Pipeline, as part of the neuronal network architecture. The flow illustrates all the stages of the pipeline, starting from the data lookup in the synapse RAM for the synaptic weight of the neuron connection. Then, a read request is sent to the stimulus RAM, which returns the current accumulated stimulus value through a system of multiple MUXes. This system represents the Read-After-Write hazard mitigation and passes the stimulus value with newly added synaptic weight instead of the old value from the stimulus RAM, in case the next neuron(s) is identical to the current processed one. Finally, the new value is also written to the stimulus RAM for use by the Neuron Core in the next simulation timestep.

6.5. Spike Overflow

As introduced at the beginning of this chapter, neuron-state updates execute deterministically, whereas spike routing is event-driven. Under high network activity, this variable routing workload may exceed the available processing budget within one simulation timestep.

To preserve the fixed execution schedule, the architecture allows the spike router to continue processing queued spikes after the neuron-core pipeline advances to the next timestep. This avoids stalling

the neuron-core pipeline until all spikes are processed, which would otherwise make the simulation timestep duration dependent on network activity and disrupt deterministic throughput.

The implemented design, therefore, prioritises fixed-rate execution over strict timestep synchronisation of synaptic updates. If the spike router workload exceeds the available processing budget, pending synaptic updates are applied in a later timestep, introducing a controlled temporal offset. Such delays are not inherently incompatible with biological operation, as synaptic transmission itself involves delays and explicit synaptic-delay mechanisms have been implemented in FPGA-based SNN applications [69]. However, the impact depends on how frequently overflow occurs. Since the simulation timestep is only 0.01 ms, occasional delays may remain negligible compared to neuronal dynamics. Therefore, the spike-overflow probability is analysed to quantify the frequency of this condition under the intended operating regime.

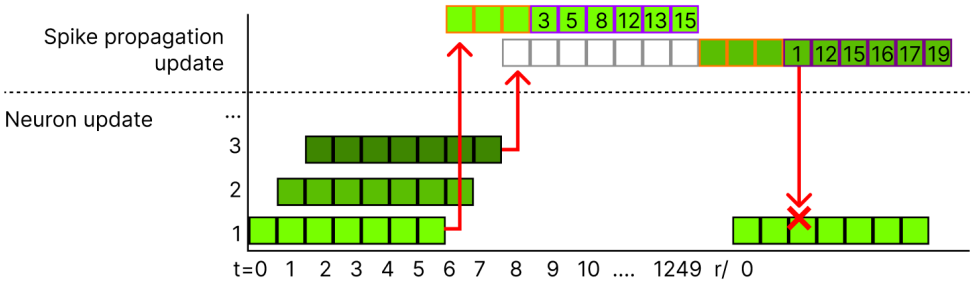


Figure 6.8: Simplistic example of a synaptic update overflowing the current simulation timestep

The maximum available processing budget per simulation timestep is determined by:

$$C_{dt} = f_{clk}\Delta t$$

With a clock frequency of 125 MHz and a timestep of 0.01 ms, this corresponds to 1.250 clock cycles per timestep. Overflow occurs when the required synaptic routing workload exceeds this budget.

Assuming neurons fire independently with average firing rate f, the probability that an individual neuron fires during a single simulation timestep is approximately $f \cdot \Delta t$. Since each of the N neurons either spikes or does not spike within that timestep, the total number of generated spikes follows a binomial distribution:

$$S \sim Binomial(N, f\Delta t)$$

Each spike requires a fixed routing overhead L, which stands for the three cycles needed for the spike FSM, and generates updates to an average of pN connected neurons, where p represents the connectivity density. Since the synaptic updates are pipelined, pN does not have to be multiplied by the number of cycles needed to update a target neuron's stimulus accumulation. The total workload becomes:

$$W = S(L + pN)$$

The expected workload, therefore, provides an estimate of the average router utilisation:

$$E[W] = Nf\Delta t(L + pN),$$

However, because spike generation is stochastic, the mean workload alone does not capture occasional overload events. Therefore, a Monte Carlo analysis is performed to estimate the probability of overflow.

The analysis evaluates network sizes and connectivity densities under two firing-rate conditions: an average activity level of 50 Hz and a high-activity case of 300 Hz. For each configuration, 2.000 time steps are sampled, and the fraction of cases where $(W > C_{dt})$ is measured. The resulting overflow probability is shown in Figure 6.9. The selected design point (1.250 neurons, 10% connectivity) remains within the router's capacity under the evaluated conditions, while the figure highlights the scaling limitations as network size or connectivity density increase.

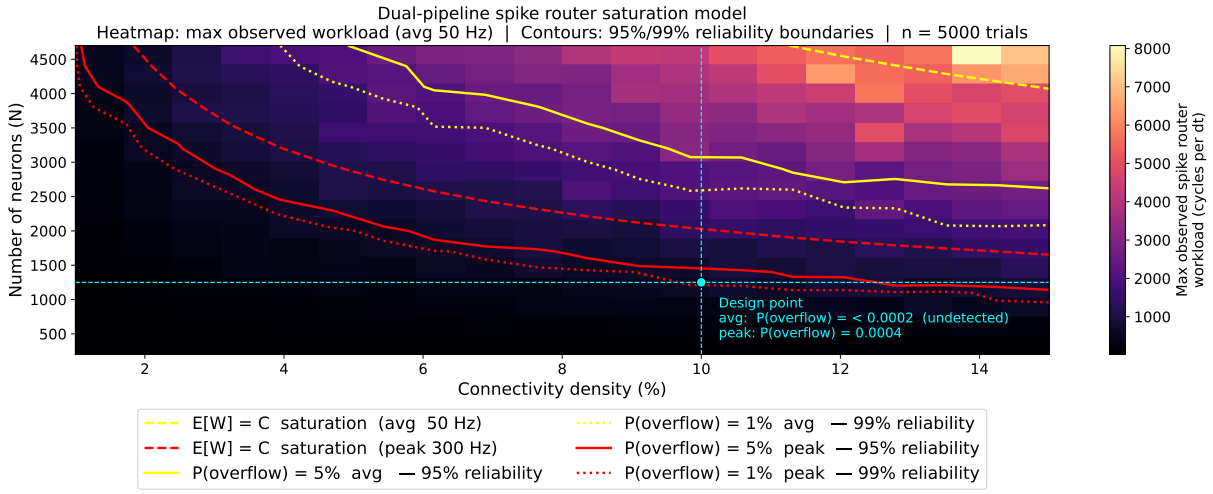


Figure 6.9: Monte Carlo estimation of spike router overflow probability for varying network sizes and connectivity densities. The colour map shows the logarithm of the probability that the required routing workload exceeds the available 1.250 cycles per simulation timestep. Solid and dotted contours indicate 95% and 99% reliability boundaries, respectively. The selected design point (1.250 neurons, 10% connectivity) is highlighted.

The evaluated design point remains within the router capacity for both the average firing-rate scenario (50 Hz) and the high-activity scenario (300 Hz), with the estimated overflow probability remaining below the plotted reliability boundaries. This indicates that the current architecture provides sufficient capacity for the intended network configuration. As network size or connectivity density increases, overflow becomes more frequent due to the increased synaptic workload. However, occasional overflow events do not directly violate the fixed-rate execution model, as queued synaptic updates are delayed by only a limited number of simulation timesteps. The analysis, therefore, identifies the operating range in which this trade-off remains acceptable and highlights the scaling limitations of the current fixed-cycle implementation.

7

Evaluation

This chapter presents the simulation and evaluation results of the proposed architectures across two levels of analysis. First, the neuron-core results are presented in Section 7.2, evaluating the cumulative effect of each optimisation step introduced in Chapter 4. This includes the progression from the baseline implementation through LUT compression, delta prediction, NeuraLUT-based approximation, and reduced-resolution configurations. Second, system-level results are presented in Section 7.3, where the most feasible, optimised neuron-core designs are integrated into the neuronal-network wrapper and evaluated under the constraints of the neuron-core wrapper. Finally, a comparative assessment against related works is provided in Section 7.4, highlighting the position of the proposed architectures within the existing FPGA-based Hodgkin-Huxley model approaches.

7.1. Experimental Setup

The evaluation was performed at three levels. Software simulations were used to assess the numerical accuracy and long-term stability of the proposed neuron-core implementations by comparing them against a high-precision Hodgkin-Huxley reference model under identical stimuli. Functional RTL simulations in QuestaSim were used to verify the correctness of the generated hardware descriptions before synthesis. FPGA resource utilisation and timing performance were obtained from Vivado post-synthesis implementation reports. Unless stated otherwise, spiking-behaviour figures and prediction metrics are derived from software simulations or hardware LUTs tested in software, whereas FPGA resource and timing results are obtained from post-synthesis reports. The fixed toolchain is summarised in Table 7.1.

Unless stated otherwise, all software evaluations used identical simulation settings. The same stimulus dataset was applied to both the reference and approximated models, while long-term behaviour was evaluated using autoregressive simulations in which each predicted state was recursively fed back as the next simulation input. For the Neural Network-based evaluations, a shorter test dataset was used, unless otherwise specified. The adapted NeuraLUT and ReducedLUT implementations, together with the code used for architecture generation and optimisation evaluation, are available in NCL's public GitHub repository [70]

FPGA synthesis was performed primarily for the Xilinx Artix-7 XC7A200T (28 nm) device, selected to provide sufficient implementation headroom. To evaluate deployment on lower-resource hardware, selected designs were additionally synthesised for the Artix-7 XC7A35T, while the final neuronal-network implementations were also evaluated on an FPGA with a different process node, namely the Artix UltraScale+ XCAU10P (16 nm), to assess the impact of a different FPGA architecture on achievable performance.

Category	Tool	Version	Purpose
Simulation	Python	3.12.3	Dataset and RTL generation
NN training	Brevitas	v0.12.1	Quantisation-Aware Training in PyTorch
NN-LUT toolchain	NeuraLUT	v2.0.0	LUT generation, in Python
LUT compression	ReducedLUT	v1.0.1	LUT optimisation, in Python
HDL simulation	QuestaSim	2021.3	RTL simulation
Synthesis	Vivado	2025.2	FPGA synthesis

Table 7.1: Toolchain used for simulation, synthesis, and optimisation

7.2. Neuron-Core Results

The neuron-core results are presented in accordance with the sequential-optimisation steps introduced in Chapter 4. Each optimisation is evaluated individually to quantify its contribution to the overall design improvement. The progression starts from the baseline architecture, and continues with LUT compression, delta prediction, Neural Network-based approximation, and reduced-resolution configurations.

7.2.1. Optimisation 1: LUT Compression

Compression results

The first objective of this optimisation was to determine whether the adapted ReducedLUT tool can significantly reduce the logic usage requirements of the gate-channel LUTs. Since smaller LUTs use less logic, the first impression of the impact of compression can be observed by comparing the bit count before and after compression, as well as the generated RTL file sizes in Verilog.

	n-gate	**m-gate**	**h-gate**
Initial LUT size [bit]	201,326,592	201,326,592	201,326,592
Compressed LUT size [bit]	3,947,849	8,825,634	3,388,101
Reduction [%]	98.04	95.62	98.32

Table 7.2: Memory reduction achieved by the adapted ReducedLUT compression algorithm for the gate-channel LUTs

For all three gate channels, the compression algorithm reduced the LUT memory requirements by more than 95%. The largest reduction was obtained for the h-gate LUT, while the m-gate LUT remained the most difficult to compress. Nevertheless, even the least compression exceeded 95%, demonstrating that a large fraction of the original state space is indeed redundant.

While the compression ratios provide insight into the reduction in lookup table size, the practical motivation for this optimisation was the inability to process the generated HDL representations within the available workflow. Therefore, the sizes of the generated Verilog files were also evaluated.

	n-gate	**m-gate**	**h-gate**
Verilog file size before compression [MB]	450.0	450.0	450.0
Verilog file size after compression [MB]	5.2	12.6	3.9
Reduction [%]	98.85	97.20	99.14

Table 7.3: Generated RTL file sizes before and after LUT compression

The reduction in LUT storage ensured substantially smaller RTL tables. The generated Verilog files decreased from several hundred megabytes to roughly two orders of magnitude less. This reduction was necessary because the uncompressed implementations exceeded the practical limits of the simulation and synthesis workflow, which prevented meaningful hardware evaluation. Since the files are now feasible for synthesis, the results are shown in Table 7.4.

	n-gate	m-gate	h-gate
Logic-Block Count	16,822	69,202	13,742
Utilisation [% of total Logic]	80.88	332.70	66.07

Table 7.4: Logic Blocks Count of each gate-channel LUT after LUT compression and individual synthesis on an Artix-7 FPGA device with 20,800 Logic Blocks available

The generated RTL became manageable after compression, yet synthesis results still require a large number of FPGA LUT blocks. Gate 'm' channel alone requires more than 300% of the LUT resources available on a low-cost FPGA such as the Artix-7. While the gate modules were synthesised individually, their combined resource usage represents an upper bound, since the complete design can reuse overlapping logic across modules. Even accounting for this potential reduction in logic utilisation, the total logic requirement is expected to remain too large for a low-resource FPGA implementation. Consequently, LUT compression alone cannot fully resolve the hardware feasibility challenges, motivating the exploration of additional optimisation strategies.

Simulation Results

As described in Chapter 4, the ReducedLUT algorithm constructs a compressed representation of the gate-channel LUTs by marking rarely activated regions as redundant, based on the generated stimulus dataset. To ensure the design remains accurate when applied to stimuli outside the original dataset, a 50-value margin around the dataset's values is also considered significant. Consequently, no major changes in the stability of the neuron dynamics are expected. The simulation results of Designs 4 [HYBRID] are shown in Figure 7.1.

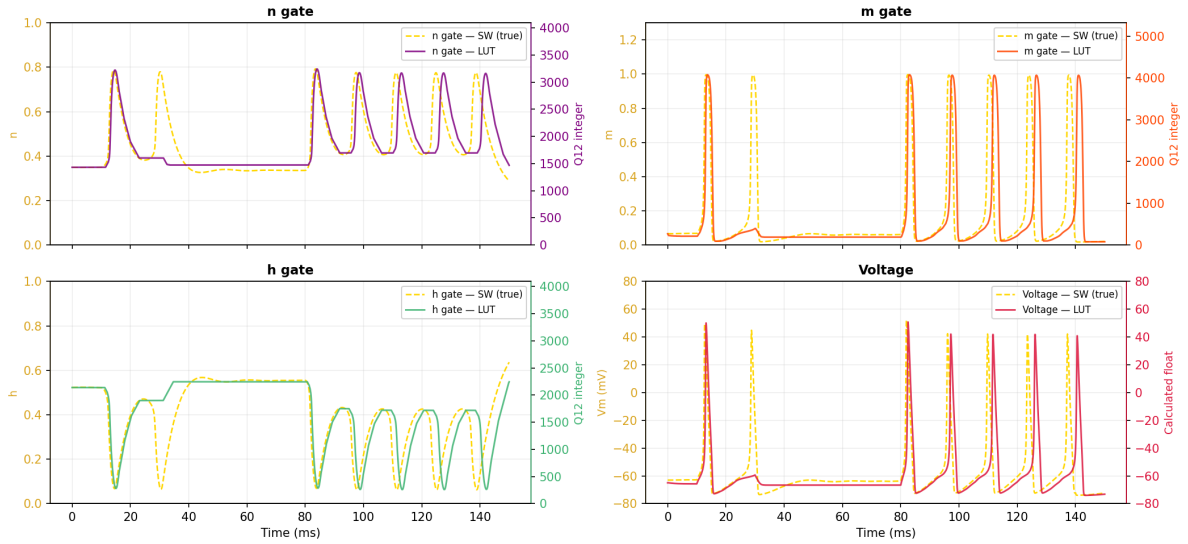


Figure 7.1: Simulation results of Design 4 [HYBRID] with LUT compression evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

The compressed gate-channel LUTs preserve the overall behaviour of the Hodgkin-Huxley dynamics, with the membrane potential trajectory and spike generation remaining mostly consistent with the expected response. However, deviations become visible in the spike timing. In particular, the second voltage spike present in the reference simulation is not reproduced by the current implementation.

To also test the compressed results of design 3 [IONIC-SUM] to see how the gate-channel LUTs behave in collaboration with compressed ionic-current LUTs, the simulation was run with an 8-bit resolution per input variable for the Sodium channel. This would affect the accuracy, but at least it would give some first impression. The results are shown in Figure 7.2.

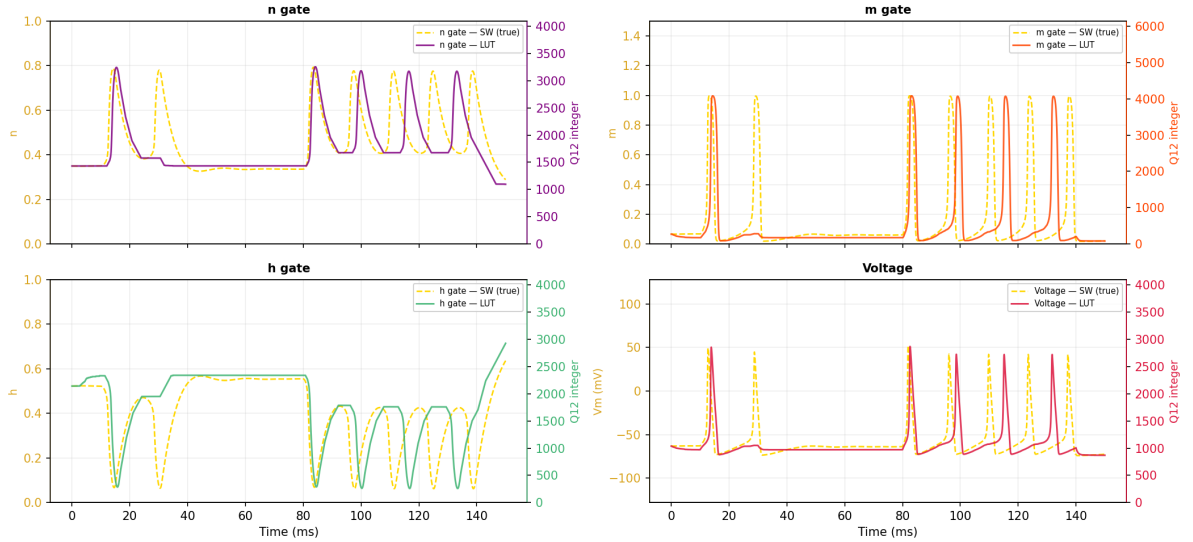


Figure 7.2: Simulation results of Design 3 [IONIC-SUM] with LUT compression evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

Compared to Design 4, the deviations in the voltage traces become clearer. Although the overall spiking behaviour remains recognisable, the reduced precision introduces additional numerical derivatives that may affect the voltage trajectory. Overall, the first optimisation demonstrates that substantial reductions in LUT storage and HDL complexity are achievable without modifying internal formulations. However, synthesis results indicate that memory compression alone is insufficient to achieve the desired logic-block count for low-resource FPGAs. Additionally, the simulation results show deviations in the neuron dynamics, including a missed spike and small temporal shifts in the voltage response. These deviations indicate a measurable impact on long-term stability, although they cannot be directly attributed to the LUT Compression algorithm alone. No baseline plot is available for comparison because the large uncompressed LUT blocks are infeasible. Hence, it is not possible to determine whether these inaccuracies were introduced by the compression process or were already present due to the quantisation mapping for next-state formulation. This motivates the investigation of additional optimisation strategies. Additionally, either due to compression or to the next-state formulation, the simulated neuron sometimes misses a stimulus peak that should have occurred. Moreover, a small temporal delay can be observed during a prolonged active stimulus period. These conclusions regarding the reduction in the logic and the non-ideal stability levels are reflected in the metrics evaluation in Figure 4.14.

7.2.2. Optimisation 2: Delta Prediction

The second optimisation investigated whether predicting state derivatives rather than absolute state values improves the numerical behaviour of the neuron updates by reducing quantisation effects during Euler propagation.

Compression results

After applying delta prediction, the generated LUTs were again processed using the adapted ReducedLUT compression algorithm. The resulting memory reductions are shown in Table 7.5.

	n-gate	m-gate	h-gate
Initial LUT size [bit]	201,326,592	201,326,592	201,326,592
Compressed LUT size [bit]	12,531,808	11,990,250	10,678,055
Reduction [%]	93.77	94.04	94.69

Table 7.5: Memory reduction achieved after applying delta prediction optimisation and ReducedLUT compression

Compared to the original compression results in Table 7.2, the achieved memory reductions remain substantial, although the relative improvement is smaller due to the delta-prediction implementation. This reduction in compression effectiveness is more clearly reflected in the Verilog file sizes generated, as shown in Table 7.6.

	n-gate	m-gate	h-gate
Verilog file size [MB]	24.69	25.27	18.15

Table 7.6: Generated Verilog file sizes for gate-channel LUTs after applying delta prediction optimisation and ReducedLUT compression

Although this reduction is slightly smaller than the compression achieved without delta prediction, the resulting LUTs represent a different trade-off: higher stability at the cost of higher logic usage.

Although this reduction is slightly smaller than the compression achieved without delta prediction, the resulting LUTs represent a different trade-off: derivative prediction reduces the direct quantisation error of the states, promising higher long-term stability, but introduces additional logic usage within the LUTs and additional memory requirements for state storage within the neuron-core wrapper. The latter is not evaluated in this section, however, since the focus lies on the neuron core requirements, and will be discussed later in the neuron-core wrapper requirements.

The increase in logic cost could be attributed to fewer opportunities for subtable matching via the ReducedLUT algorithm. For visual aid, the output values of the LUTs, the derivative representations of the gate channels, in relation to the membrane voltage and with time, are plotted in Figure 7.3. As can be seen, the range of the gate m channel derivative is much higher than that of the n and h channels. For consistency and simplicity when denormalising the outputs in the neuron core, a normalisation range of -1 to +1 is chosen. Denormalisation can be easily performed with shift operations, and the gate m channel doesn't suffer from prediction errors due to truncation, because only very few values fall below -1 or above +1. The reconstructed gate m channel plot from quantised derivative LUT outputs, along with its maintained accuracy compared with the ground truth, is shown in the Appendix Section B.2.

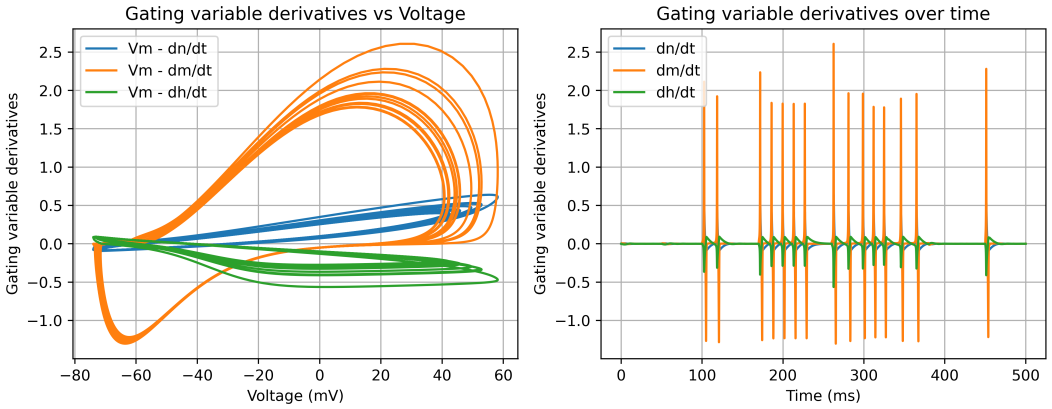


Figure 7.3: Simulated representation of the relation between the gate channel derivatives and the membrane voltage (left) and the gate-channel behaviour across 500 ms with various stimuli applied (right)

This optimisation primarily targets the representation of small state changes during Euler propagation and the numerical stability of the state evolution, rather than reducing LUT size. Even without delta prediction, additional optimisations had to be investigated to reduce logic resource usage, so this optimisation is kept as a necessary intermediate step before the next compression optimisations.

Simulation Results

The effectiveness of the delta prediction in long-term stability is tested on both Design 4 [HYBRID], shown in Figure 7.4, and Design 3 [IONIC-SUM], shown in Figure 7.5.

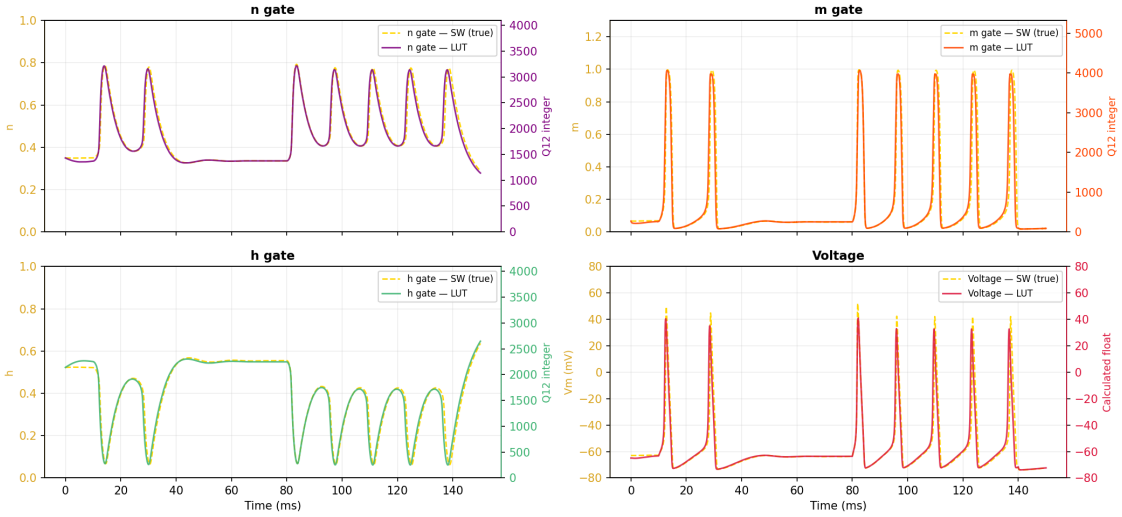


Figure 7.4: Simulation results of Design 4 [HYBRID] with LUT compression and delta prediction evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

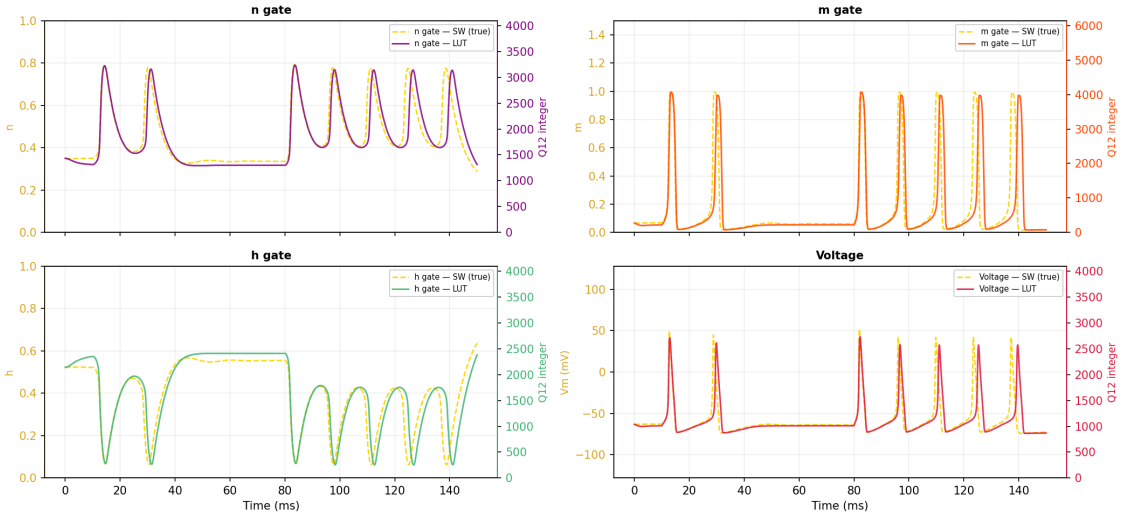


Figure 7.5: Simulation results of Design 3 [IONIC-SUM] with LUT compression and delta prediction evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

The results indicate that delta prediction preserves the original neuronal dynamics more accurately than the approach that uses the first optimisation step. Nevertheless, Design 3 [IONIC-SUM] still exhibits temporal delays in Figure 7.5 during prolonged active stimulus periods, particularly around the spikes occurring at roughly 115, 125, and 135 ms. However, the differences are less substantial than after the first optimisation in Figure 7.2. Since Design 4 no longer shows any temporal deviations, the deviations observed in Design 3 are now primarily attributed to the reduced bit resolution in the Sodium LUT introduced for feasibility rather than for optimisation.

Metrics Evaluation

The evaluation indicates that delta prediction reduces the approximation error introduced by quantisation, thereby improving the accuracy of neuron dynamics and supporting long-term stability. However, the results in Table 7.5 show a slight increase in logic utilisation compared to the first optimisation step

in Table 7.2. Therefore, delta prediction should be considered as a supporting technique rather than a standalone solution for reducing hardware complexity. To conclude, the optimisation trend shown in Figure 4.14 must be adjusted, as the introduction of delta prediction results in a small increase in logic usage rather than maintaining approximately constant logic utilisation compared with the prior optimisation step.

7.2.3. Optimisation 3: NeuraLUT

The third optimisation investigated whether a Neural Network-based LUT approach could replace the large discrete lookup tables. The optimisation replaces large-dimensional and/or complex LUT mappings with a collection of simpler LUTs representing the individual perceptrons, each with its own learned weights and activations.

The evaluation is performed on multiple architectures. Design 4 [HYBRID] remains the primary reference because it represented the most feasible synthesis implementation during the design exploration. Design 1 [MONOLITH] is also evaluated to investigate whether NeuraLUT can overcome the dimensionality bottleneck of the original LUT approach. Furthermore, control experiments without delta prediction are included in the results to distinguish the effect of the neural approximation from the effect of the derivative-based optimisation.

In all cases, the simulation results are presented first. Whenever the generated RTL is sufficiently small to be processed by the LUT generation, compression, and synthesis workflow, the corresponding compression and synthesis results are presented afterwards.

Design 4 [HYBRID]

The first evaluated configuration applies NeuraLUT to the gate-channel LUTs while keeping the conductance-current calculations unchanged, since these are either 1-dimensional LUTs or multiplications. Since with Euler propagation, the second part of the Delta-prediction optimisation, the old value is used to calculate the new value, such a system is called an autoregressive system. The autoregressive simulation results are shown in Figure 7.6. The prior neural network training results are summarised in Table 7.7 and shown in detailed plots in Section B.4.1.

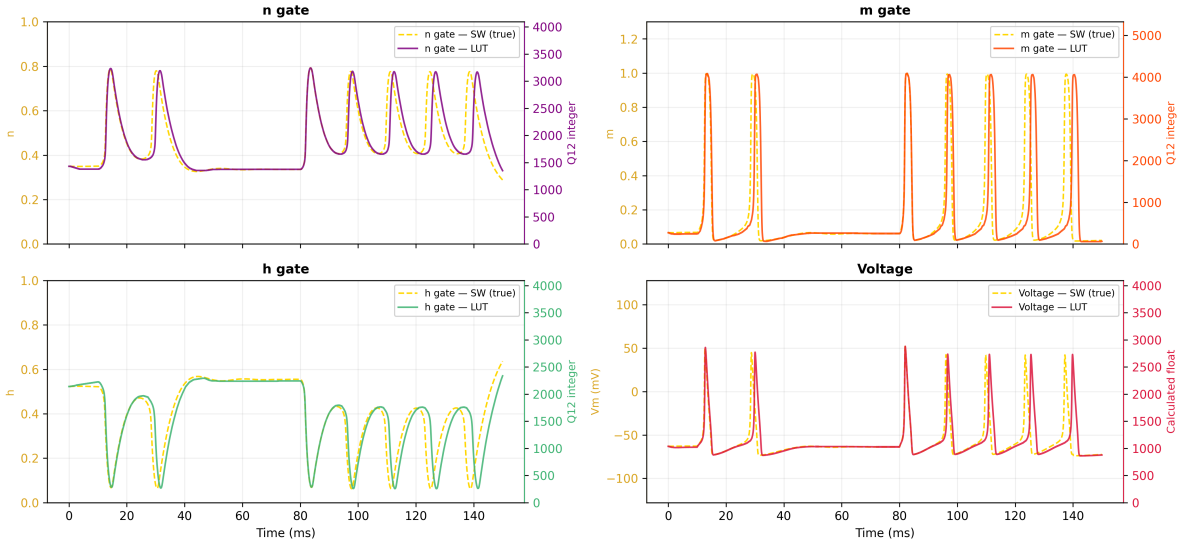


Figure 7.6: Autoregressive simulation results of Design 4 [HYBRID]with NeuraLUT-based gate-channel LUTs evaluated in software. The gate-channel LUTs are built with an MLP configuration of 3 input-, 2 hidden-, and 1 output-layer perceptrons. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

Variable	MSE	RMSE	R^2
n gate	0.0017	0.0407	0.9916
m gate	0.0214	0.1464	0.8910
h gate	0.0001	0.0118	0.9992

Table 7.7: Summarised autoregressive Neural Network prediction statistics per gate-channel. The calculation and interpretation of the MSE, RMSE, and R^2 metrics are described in Section 5.2. Corresponding prediction plots are provided in Section B.4.1.

The generated hardware representation consists of multiple compressed perceptron-based LUTs rather than state-transition LUTs. The resulting Verilog file sizes are summarised in Table 7.13. The values shown are the summed Verilog file sizes of the full 3-2-1 MLP configuration in each LUT. In other words, combining the sizes of the three input layer perceptrons, 2 hidden layer perceptrons, and finally one output layer perceptron.

	n-gate	**m-gate**	**h-gate**
Total Verilog size [MB]*	99.7	19.0	115.4

Table 7.8: Generated Verilog sizes for each NeuraLUT-based gate-channel implementation, where all the corresponding perceptron-based LUT sizes are summed together to show the total result

Compared with the compressed LUT implementation before the NeuraLUT approach, the generated RTL files are larger, as expected. This approach nevertheless prevents the exponential scaling that can arise from some potentially high-dimensional LUTs, such as the sodium current LUT. It also provides an opportunity to reduce bit resolution for specific configurations discussed in Chapter 4. At the current RTL file sizes, however, synthesis cannot be performed to test the optimisation against the hardware design metrics. Further reduction optimisations are therefore required before applying the tool to Designs 3 and 2, which include the gate channel LUTs.

Design 1 [MONOLITH]

To investigate whether NeuraLUT can address the dimensionality limitation of the monolithic architecture, a full (derivative) neuron-state approximation was evaluated. As a recap, the full state-transition mapping becomes:

$$(V_m, n, m, h, I) \rightarrow (\frac{dV_{next}}{dt}, \frac{dn_{next}}{dt}, \frac{dm_{next}}{dt}, \frac{dh_{next}}{dt}),$$

using Neural Network-derived LUTs. The most 'accurate' autoregressive prediction results are shown in Figure 7.7. In addition to the usual n,m,h and V-plots, the limit cycles V-n, V-m, and V-h are also included in the figure for a more detailed observation.

The obtained autoregressive neural network metrics are shown in Table 7.9.

Variable	MSE	RMSE	R^2
Voltage	1169.9833	34.2050	-2.3340
n gate	0.0109	0.1046	0.3464
m gate	0.0949	0.3081	-1.0093
h gate	0.2361	0.4860	-9.0071
Overall	292.6108	17.1059	-3.0011

Table 7.9: Autoregressive prediction Neural Network metrics for Design 1 [MONOLITH] with NeuraLUT

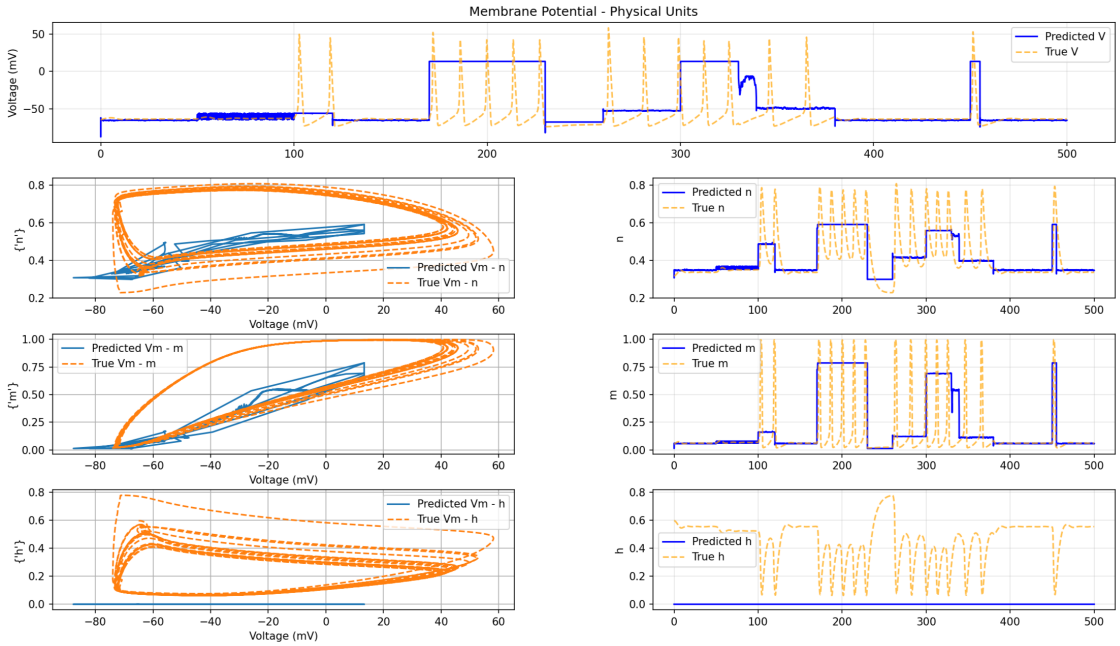


Figure 7.7: Autoregressive simulation results of Design 1 [MONOLITH] with NeuraLUT-based LUTs evaluated in software. The voltage LUT is built with an MLP configuration of 10 input-, 8 first-hidden-, 6 second-hidden-, and 4 output-layer perceptrons. The figure displays the time evolution of the 4 predicted output states, including the gating variables (n, m, and h on the right side) and the membrane voltage (V_m on the top side). Additionally, the relationships between each gate-channel variable and the membrane voltage are shown on the left. For detailed observation of the behaviour, the results are simulated on the full dataset on which it was trained rather than the shorter testing dataset from previous figures.

The results show that although the neural approximation can reconstruct some spike-like activity during the active spike period, the predictions are very far off. Also, it couldn't predict any sensible gate h value. This indicates that replacing the full LUT with a neural approximation, using the configurations and fine-tuning attempts discussed in Section 5.2, cannot sufficiently replace the dynamics of the monolithic architecture.

Control Experiment: Without Delta Prediction

To separate the influence of NeuraLUT from the derivative-based prediction and the autoregressive Euler propagation, the same designs were evaluated without the delta prediction optimisation.

The corresponding simulation results for Design 1 [MONOLITH] are shown in Figure 7.8. The voltage LUT is built with an MLP configuration of 10 input-, 8 first-hidden-, 6 second-hidden-, and 4 output-layer perceptrons. For detailed observation of the behaviour, the results are simulated on the full dataset on which it was trained, rather than on the shorter testing dataset used in previous figures.

The corresponding one-step prediction performance is shown in Figure 7.9. This evaluation assesses the neural network's local approximation capability. However, high one-step prediction accuracy does not guarantee stable autoregressive rollout behaviour, as the model's recursive implementation can amplify small prediction errors over time.

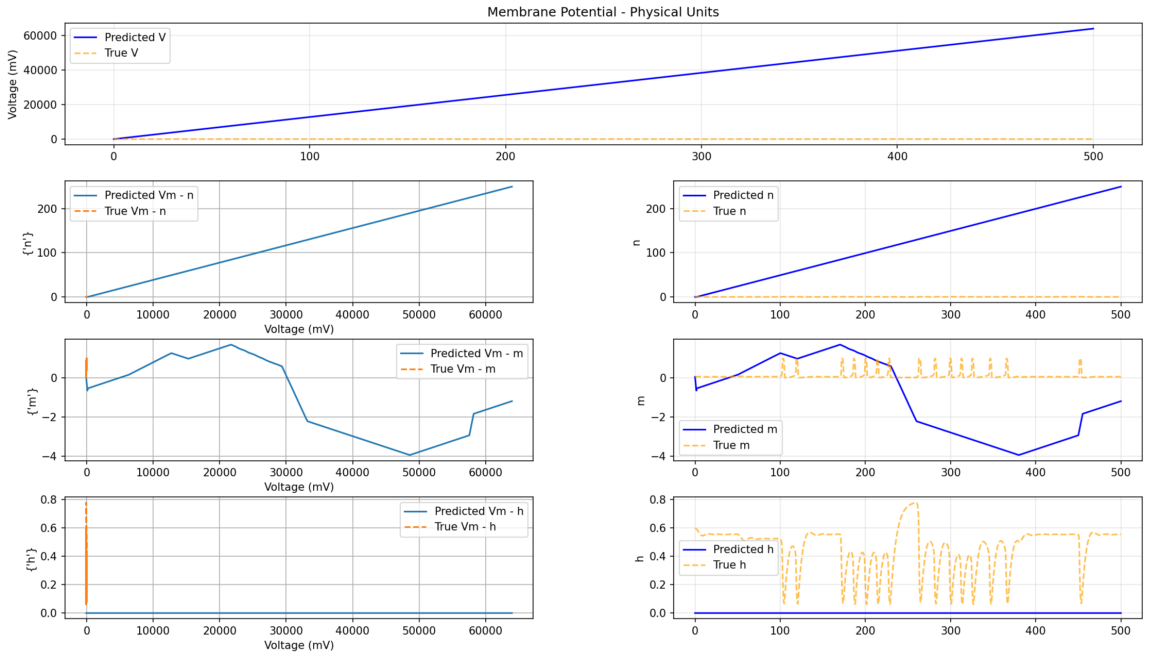


Figure 7.8: Autoregressive simulation results of Design 1 [MONOLITH] with NeuraLUT-based gate-channel LUTs evaluated in software on the full dataset, but without the delta prediction approach as a control experiment. The figure displays the time evolution of the 4 predicted output states, including the gating variables (n, m, and h on the right side) and the membrane voltage (V_m on the top side). Additionally, the relationships between each gate-channel variable and the membrane voltage are shown on the left.

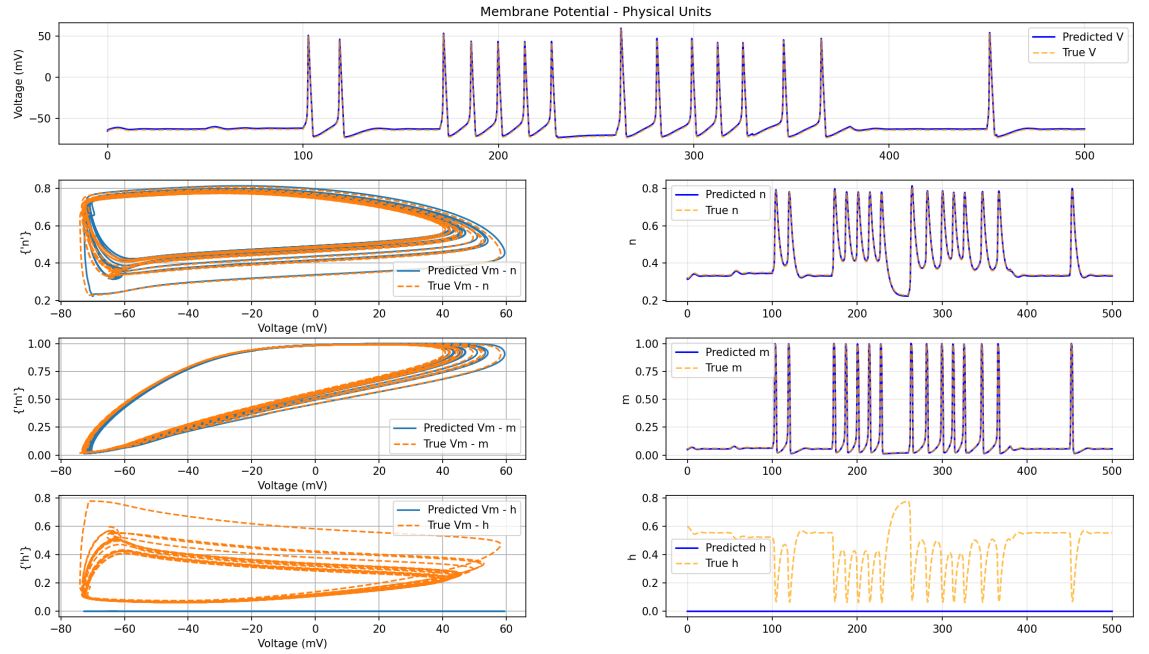


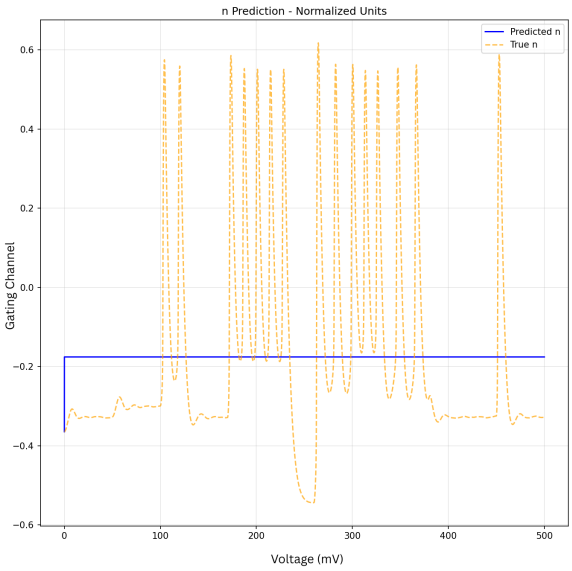
Figure 7.9: One-step-prediction simulation results of Design 1 [MONOLITH] with NeuraLUT-based gate-channel LUTs evaluated in software on the full dataset, but without the delta prediction approach as a control experiment. The figure displays the time evolution of the 4 predicted output states, including the gating variables (n, m, and h on the right side) and the membrane voltage (V_m on the top side). Additionally, the relationships between each gate-channel variable and the membrane voltage are shown on the left.

The one-step prediction results demonstrate that the neural network can accurately approximate individual transitions. However, when the model is used in an autoregressive manner, prediction errors

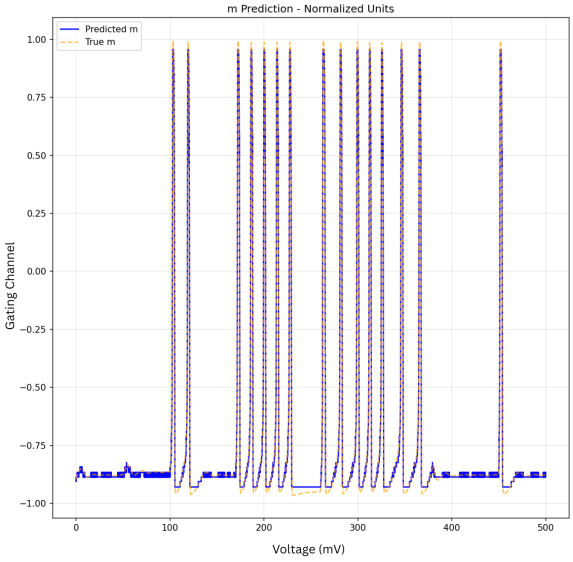
accumulate over time, resulting in increasing deviation from the expected neuron dynamics and eventual instability. Additionally, the gate 'h' prediction remains constant at zero throughout the simulation, indicating that the network fails to capture this gate's temporal behaviour. A possible explanation is that this particular input state does not provide sufficient information for the network to infer the gate's evolution, or that the training process does not adequately capture its contribution to the output prediction. Such behaviour was only observed for multi-output networks. Further investigation of the input representation and training configuration would be required to confirm the underlying cause. Since the other state variables didn't give promising results either, this design was not investigated further.

Regarding the gate-channel LUTs, which also use Euler propagation, an attempt has been made to enable the Neural Network to learn these dynamics. As can be seen in Figures 7.10a, 7.10b and 7.11a, the autoregressive simulation has failed mainly for the 'n' and 'h' channels. However, even the gate 'm' channel isn't as successful as can be perceived from a first impression of the figure. When zooming in on the subthreshold value range, aggressive oscillations are to be found.

To verify that the issue lies with removing delta prediction and not with the neural-network configuration, the same training and test sets have been applied to the ionic-current blocks, since they do not rely on derivatives or previous values for their next-state computation, but rather exclusively on the incoming inputs. First tests on these currents are shown in Figures 7.11b to 7.11d. As can be observed, they already perform significantly better than their gate-channel counterparts. This motivates the optimisation trajectory to rightfully include delta-prediction optimisation for the gate channels in the upcoming optimisation phases.

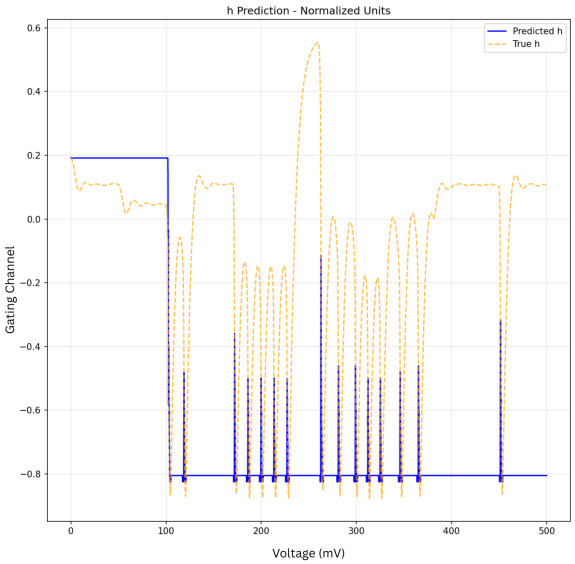


(a) Autoregressive test of the learned gate 'n' channel dynamics by NeuraLUT without delta prediction

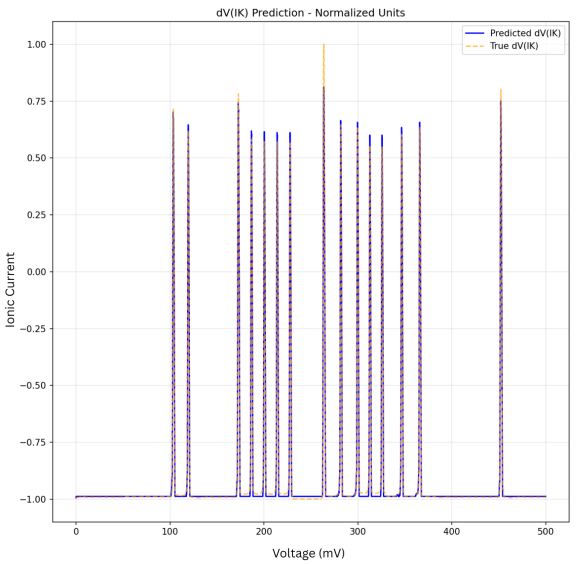


(b) Autoregressive test of the learned gate 'm' channel dynamics by NeuraLUT without delta prediction

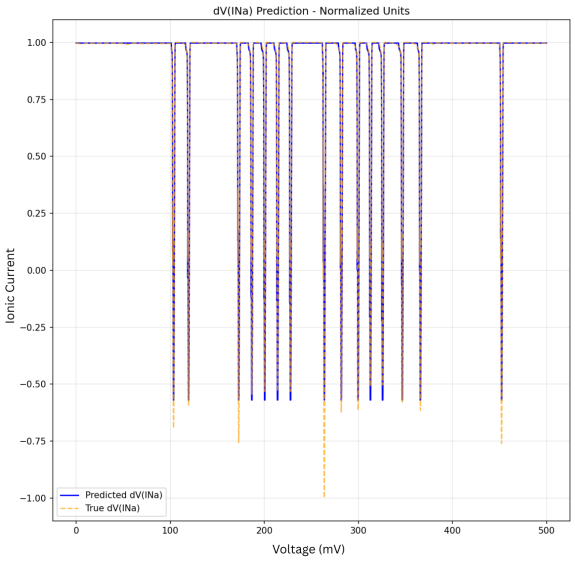
Figure 7.10: Control Experiments of Optimisation 3 (NeuraLUT) without optimisation 2 (delta prediction), part 1/2



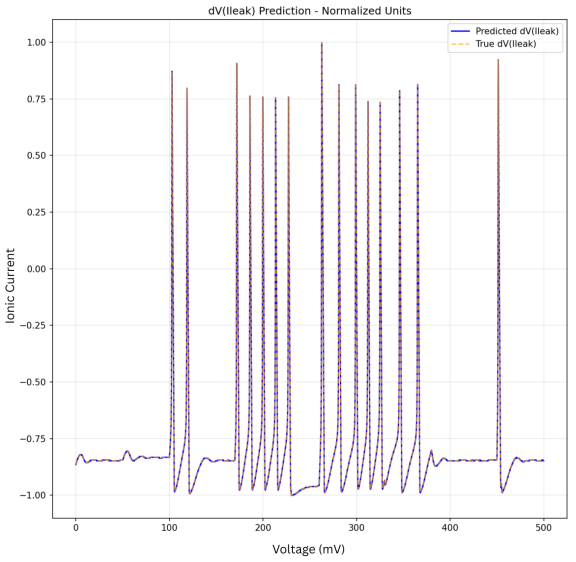
(a) Autoregressive test of the learned gate 'h' channel dynamics by NeuraLUT without delta prediction



(b) Autoregressive test of the learned I_K current dynamics by NeuraLUT without delta prediction



(c) Autoregressive test of the learned I_{Na} current dynamics by NeuraLUT without delta prediction



(d) Autoregressive test of the learned I_{leak} current dynamics by NeuraLUT without delta prediction

Figure 7.11: Control Experiments of Optimisation 3 (NeuraLUT) without optimisation 2 (delta prediction), part 2/2

Metrics Evaluation

NeuraLUT changes the hardware representation fundamentally. The original LUT entries are no longer explicitly generated for every input combination; instead, the behaviour is represented through a network of perceptron LUTs. Therefore, the resulting logic cost depends on the number of perceptrons in each layer and the precision settings.

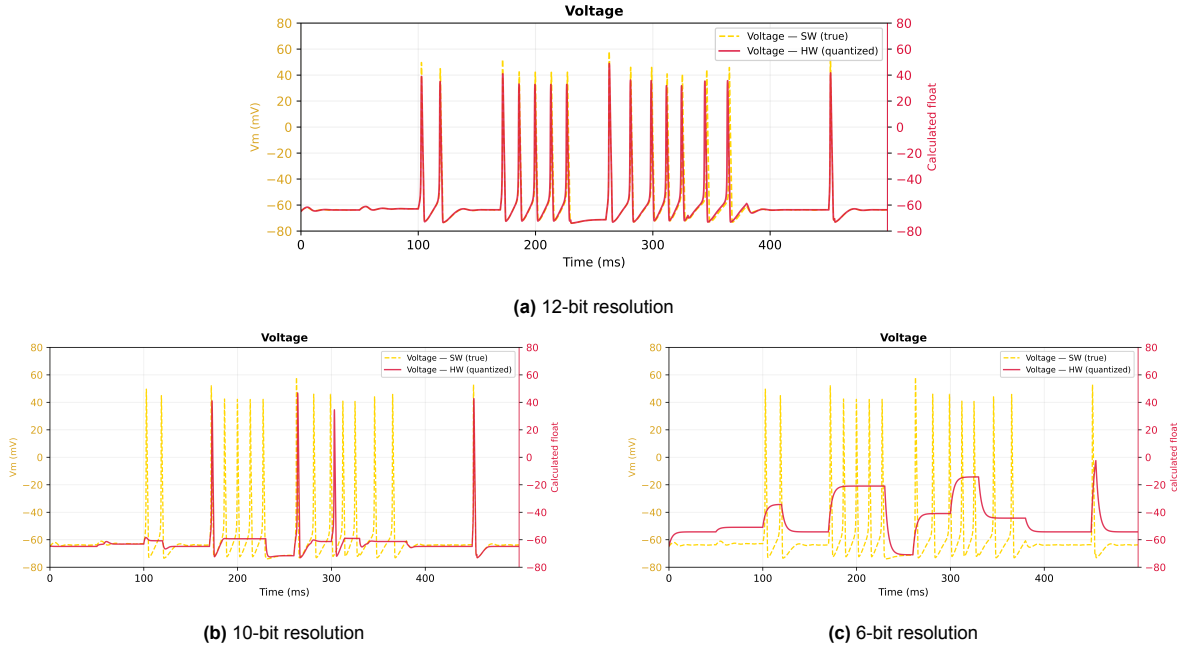
This optimisation introduces an additional approximation stage compared to the previous optimisations. As was evident from the various attempts to learn the selected architectures, stability is expected to depend strongly on the selected network configuration and training method. Even though, in the end, a stable system with three Neural Network-derived LUTs was achieved for Design 4, the strong decline in stability in Figure 4.14 may be considered less extreme. If the metrics were to reflect Design 1, the graph would be completely different.

7.2.4. Optimisation 4: Bit-Resolution Reduction

The final optimisation investigates whether the precision of the NeuraLUT-generated perceptron LUTs can be reduced. The optimisation reduces the neural network's internal precision while maintaining a 12-bit output resolution in the last layer's perceptron LUTs. This preserves compatibility with the remaining neuron core architecture while reducing the size of the intermediate LUT structures.

Resolution-Reduction Only

Before applying the reduction to complete the architecture, the effect of reducing precision on the design without a neural network was evaluated first. Even though a 12-bit resolution was determined to be the minimum required to capture the Hodgkin-Huxley dynamics, as discussed in Section 3.1.1, a check with simulated LUTs at a lower resolution should still be performed. This is done for Design 4, with LUT compression and delta-prediction already applied, since this has proven to be an accurate representation. The results in Figure 7.12 demonstrate that reducing resolution directly causes significant degradation in accuracy, motivating the use of a neural-network approach to compensate for the reduced intermediate precision. A detailed comparison of the resolutions, including additionally the gate-channel plots, is shown in Section B.3.



(a) 12-bit resolution

(b) 10-bit resolution

(c) 6-bit resolution

Figure 7.12: Comparison between three simulations that illustrate the effect of a reduced bit resolution on the LUT on which only Optimisations 1 and 2 have been applied; from pane (a) to (c) comparing between 12 bits, 9 bits and 6 bits

Design 4 [HYBRID]

The HYBRID architecture was evaluated first because it is the lowest-dimensional and most feasible one. Reduced precision is applied to the neural-network-generated gate-channel LUTs while maintaining the same architecture as in the neural-network optimisation. The final configuration that yielded accurate results used a 6(input)-6(hidden)-12(output) bit-resolution configuration within an MLP comprising 4 input, 2 hidden, and 1 output perceptrons. The individual autoregressive test results of each gate channel are presented in Figures B.11 to B.13 in Appendix Section B.4.2, while a statistical summary of these results can be found in Table 7.10.

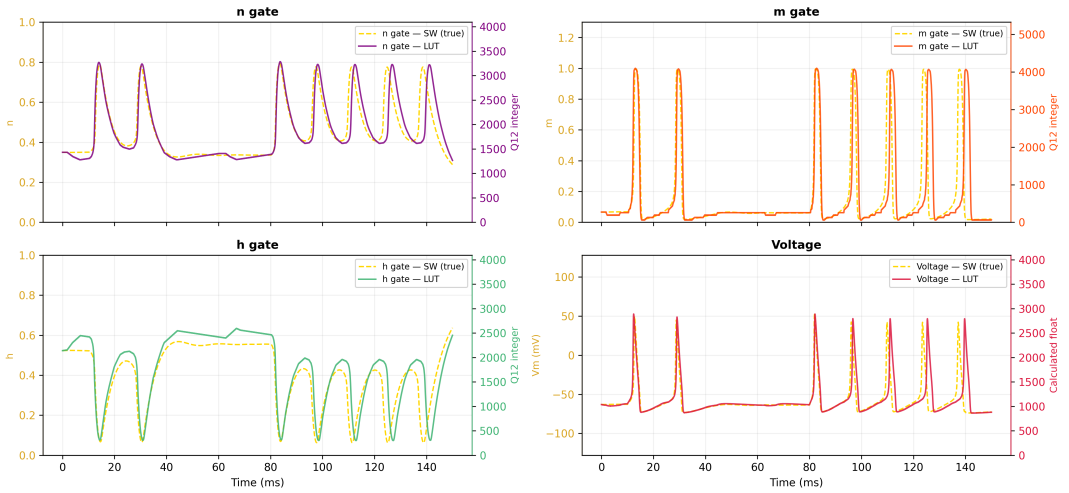


Figure 7.13: Autoregressive simulation results of Design 4 [HYBRID] with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

Variable	MSE	RMSE	R^2
n gate	0.0001	0.0100	0.9985
m gate	0.0187	0.1368	0.9010
h gate	0.0003	0.0186	0.9963

Table 7.10: Summarised autoregressive Neural Network prediction statistics per gate-channel with reduced input and hidden resolution. The corresponding plots can be found in Section B.4.2

.

The results demonstrate that the neural networks generated by NeuraLUT can capture the complex dynamics of the state variables with lower resolution and still accurately reconstruct the next states using their hidden weights and activation functions. The corresponding synthesis results in Vivado are summarised in Table 7.11.

Unlike the previous evaluations, this synthesis is performed on the complete integrated system, including the Neuron-Core Wrapper and Spike Router, as described in Chapter 6. Consequently, the reported resource utilisation reflects the final hardware implementation, in which Vivado can optimise resource sharing between interconnected modules. For example, logic structures or memory resources may be reused when synthesis identifies equivalent functionality.

An additional observation is that Vivado mapped part of the gate m-LUT implementation to two BRAM-18 blocks, rather than implementing the complete structure with logic resources, as it has done for the gate n and gate h-LUTs. This resulted in more efficient resource distribution without increasing the critical path delay, a metric that identifies the longest path through which a signal must be processed on time. This behaviour is likely caused by the synthesis and placement-optimisation process, in which the availability and physical location of BRAM resources enable a memory-based implementation without adversely affecting timing performance. Regarding the reporting of these resources in Table 7.11, BRAM-18k utilisation is not reported separately since BRAM-18k and BRAM-36k blocks share the same physical memory tiles.

Module / Instance	LUTs	BRAM-36k	BRAM-18k	DSP
gate n-LUT	1,576	0	0	0
gate m-LUT	801	0	2	0
gate h-LUT	1,675	0	0	0
$\bar{g}_K n^4$-LUT	714	0	0	0
$\bar{g}_{Na} m^3$-LUT	895	0	0	0
I_{leak}-LUT	64	0	0	0
Neuron Core Total	6,154	0	2	4
Neuron-Core Wrapper Total	6,298	6	8	4
Top-Level Total	6,307	7	8	4
XC7A200T Capacity	133,800	365	730	740
FPGA Utilisation [%]	4.71	3.01	-	0.54
XC7A35T Capacity	20,800	50	100	90
FPGA Utilisation [%]	30.34	22.00	-	4.44

Table 7.11: FPGA Resource Utilisation Summary by synthesising the full integrated system corresponding to Design 4[HYBRID] on mid-range (XC7A200T) and a low-end (XC7A35T) FPGA device

Design 3 [IONIC-SUM]

Design 3 extends the reduced-resolution evaluation by applying the optimisation to both the gate-channel and ionic-current approximations. The individual autoregressive evaluations of the ionic currents I_K and I_{Na} are provided in Appendix Section B.4.3 in Figures B.14 and B.15. The statistical summary of these Neural Network-derived LUTs during autoregressive evaluation is given in Table 7.12. The experiments show that ionic-current neural networks require a higher input resolution than gate-channel neural networks to maintain accurate autoregressive behaviour. Whereas the gate-channel approximations remain stable with a 6-bit input and hidden-layer resolution, the ionic currents require an 8-bit input resolution and a 6-bit hidden-layer resolution.

This difference is likely related to both the numerical representations of the functions and the architecture configuration. Ionic currents operate over a substantially larger value range than the gate variables, spanning approximately -12 to 0, or 0 to +12, compared to the 0-to-1 range of the gate variables. Furthermore, the gate channels are built on derivative-based dynamics, whereas the ionic current directly approximates its next state, resulting in different sensitivities to quantisation errors.

Variable	MSE	RMSE	R^2
IK	0.0000	0.0048	0.9995
INa	0.0001	0.0103	0.9974

Table 7.12: Summarised autoregressive Neural Network prediction statistics for the Potassium and Sodium currents with reduced input (8b) and hidden (6b) resolution. The corresponding plots can be found in Section B.4.3.

	I_K	I_{Na}	I_{leak}
Total Verilog size [MB]*	3.72	4.53	0.03
Total Logic Usage [FPGA LUTs]	3555	4643	52

Table 7.13: Generated Verilog sizes and Vivado synthesis logic usage for each ionic current LUT implementation after all optimisations, where all the corresponding perceptron-based LUT sizes are summed together to show the total result.

For the ionic leakage current I_{leak}, the original one-dimensional LUT from prior optimisation stages is used, as this component has already been fully simplified. Combining the resulting LUT components within the proposed Design 3 architecture produces the autoregressive rollout shown in Figure 7.14. Bit resolution is applied in a 6b(input)-6b(hidden)-12b(output) configuration for the gate-channel LUTs,

with an MLP configuration of 4 input-, 2 hidden-, and 1 output-layer perceptrons; and an 8b(input)-6b(hidden)-12b(output) configuration for the ionic-current-LUT with an MLP configuration of 4 input-, 2 hidden-, and 1 output-layer perceptrons.

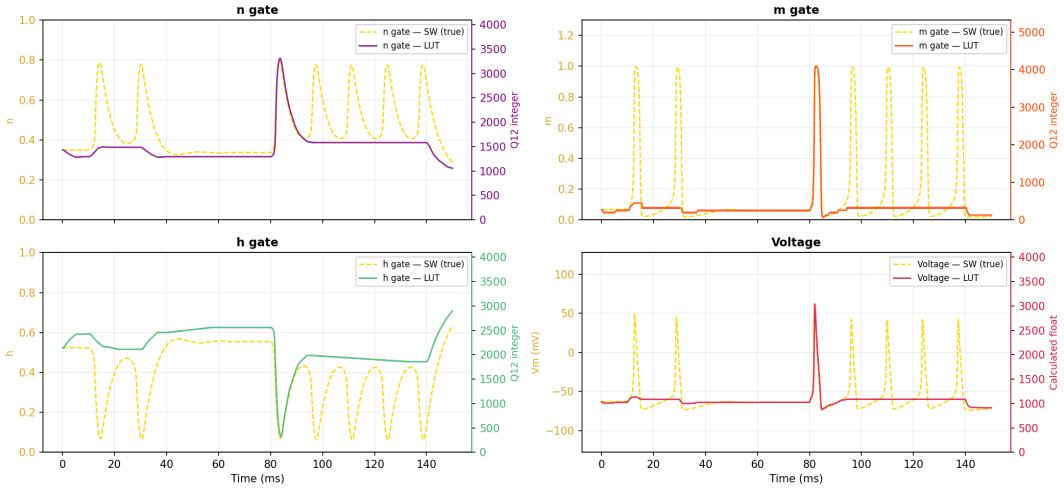


Figure 7.14: Autoregressive simulation results of Design 3 [IONIC-SUM] with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis).

Despite the individual LUTs for the gate channels and the ion currents being highly accurate, integrating all Neural Network-derived LUTs fails to capture all the stimulus spikes that should have been triggered. To examine the behaviour more closely and to provide more example cases where the system succeeds and where it fails to capture the spiking dynamics, a larger autoregressive evaluation is conducted using the same dataset used for training in Section 5.2. The resulting plot is shown in Figure 7.15. Bit resolution is applied in a 6b(input)-6b(hidden)-12b(output) configuration for the gate-channel LUTs, with an MLP configuration of 4 input-, 2 hidden-, and 1 output-layer perceptrons; and an 8b(input)-6b(hidden)-12b(output) configuration for the ionic-current-LUT with an MLP configuration of 4 input-, 2 hidden-, and 1 output-layer perceptrons.

Now, as illustrated in Figure 7.15, the autoregressive rollout shows that, for new incoming stimuli, the network often captures only the initial spike response within a series of spikes, for example, at timesteps of 180 ms and 260 ms. However, during prolonged periods of constant stimulation, the network fails to reproduce the expected repetitive spiking behaviour. For example, during the active stimulus period between 180 and 220 ms, four additional spikes would be expected after the initial response. This behaviour indicates that, although the stimulus current is not provided as an explicit network input, the resulting changes in the membrane voltage and gating variables contain sufficient information for the neural network to capture the initial state transitions following stimulus onset. The learned representation is, however, insufficient to reproduce the dynamics required for sustained repetitive spiking.

Although this limitation prevents continuous, repetitive firing under ideal constant stimulation, it does not necessarily invalidate the design for event-driven neuronal network simulations. Biological stimulation is typically represented by discrete synaptic events rather than prolonged constant currents [66]. Furthermore, the underlying hardware architecture can generate new spikes whenever incoming events drive the neuron state across the activation threshold. Therefore, the design remains suitable for integration into large-scale, event-driven simulation platforms and is selected as the secondary feasible architecture for the neuron-core wrapper implementation.

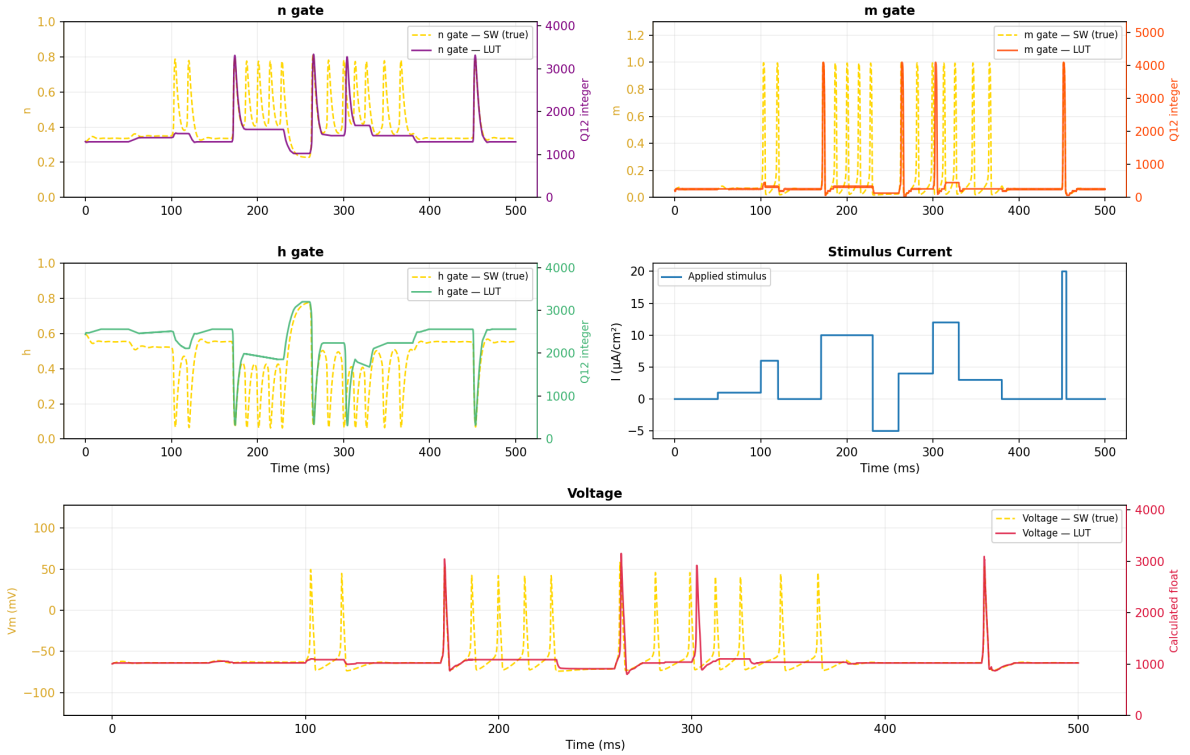


Figure 7.15: Autoregressive simulation results of Design 3 [IONIC-SUM] with NeuraLUT-based gate-channel LUTs in combination with reduced bit-resolution evaluated in software on the full training dataset for detailed observation. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). Additionally, the middle-right figure shows the stimulus injection trajectory used in the autoregressive simulation.

The corresponding synthesis results are summarised in Table 7.14. Compared to Design 4, this architecture requires approximately twice the logic utilisation while eliminating the use of dedicated DSP blocks entirely. This reflects a design trade-off between reducing specialised arithmetic resources and increased reliance on logic utilisation, while also showing a degradation in sustained dynamical behaviour, as the model no longer maintains stable repetitive spiking under prolonged stimulation.

Module / Instance	LUTs	BRAM-36k	BRAM-18k	DSP
gate n-LUT	1,357	0	0	0
gate m-LUT	726	0	2	0
gate h-LUT	881	0	0	0
I_K-LUT	3,555	1	2	0
I_{Na}-LUT	4,643	0	0	0
I_{leak}-LUT	52	0	0	0
Neuron Core Total	12,751	1	4	0
Neuron-Core Wrapper Total	12,895	7	10	0
Top-Level Total	12,905	8	13	0
XC7A200T Capacity	133,800	365	730	740
FPGA Utilisation [%]	9.64	3.97	-	0.00
XC7A35T Capacity	20,800	50	100	90
FPGA Utilisation [%]	63.49	29.00	-	0.00

Table 7.14: FPGA Resource Utilisation Summary by synthesising the full integrated system corresponding to Design 3 [IONIC-SUM] on mid-range (XC7A200T) and a low-end (XC7A35T) FPGA device

Similar to Design 4, Vivado performs additional resource optimisation during synthesis by mapping parts of the LUT components to available memory resources. The m-gate LUT is again implemented using two BRAM-18 blocks, while the I_K-LUT is additionally mapped to one BRAM-36 block and two BRAM-18 blocks. This reduces the required logic resources while maintaining the required functionality and timing constraints.

It should be noted that the resource utilisation of individual modules cannot be directly summed to obtain the final design utilisation. During full-system synthesis, Vivado considers the complete design hierarchy and can optimise resource sharing between interconnected modules. Therefore, the reported logic counts represent their contribution within the complete implementation context, rather than standalone absolute counts. This explains the difference in the sum of logic blocks across the individual gate channel LUTs between Design 3 and 4 in Table 7.14 and Table 7.11, respectively.

Design 2 [SPLIT]

Design 2 returns to the more coarse-grained structure for predicting the membrane voltage. The neural network has to learn the mapping from the five input state variables n, m, h, V, and I to the $\frac{dV}{dt}$ output prediction. The autoregressive results are shown in Figure 7.16. Again, the full dataset is used to test the design's autoregressive behaviour to gain a broader view of its actions. The summarised statistic of the individual autoregressive test of the voltage LUT is shown in Table 7.15. The autoregressive evaluation is plotted and shown in Figure B.17 in Appendix Section B.4.4. Bit resolution is applied in a 12b(input)-6b(hidden)-6b(hidden)-12b(output) configuration for the Voltage-LUT with an MLP-configuration of 8 input-, 6 first-hidden-, 4 second-hidden-, and 1 output-layer perceptrons; And a 6b(input)-6b(hidden)-12b(output) configuration for the gate-channel LUTs with an MLP-configuration of 4 input-, 2 hidden-, and 1 output-layer perceptrons.

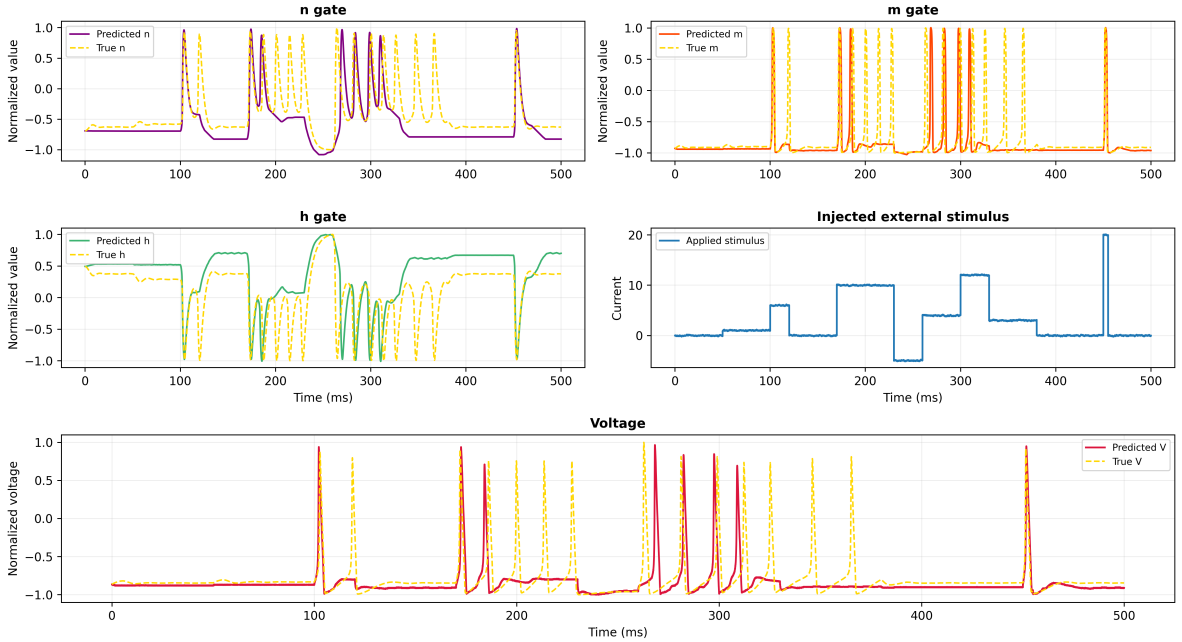


Figure 7.16: Autoregressive simulation results for Design 2 [SPLIT] with NeuraLUT-based gate-channel LUTs and reduced bit-resolution, evaluated in software on the full training dataset for detailed observation. The figure displays the time evolution of the gating variables (n, m, and h) and the resulting membrane voltage (V_m). Each subplot compares the high-precision software implementation ("SW (true)" using the left y-axis) against the LUT-based approximation ("LUT" using the right y-axis). Additionally, the middle-right figure shows the stimulus injection trajectory used in the autoregressive simulation.

Variable	MSE	RMSE	R^2
Voltage LUT	0.2513	0.5013	0.9993

Table 7.15: Summarised autoregressive Neural Network prediction statistics for the 5-input Voltage LUT for design 2 within Optimisation 4. The corresponding plots can be found in Section B.4.4.

Again, it can be observed that even though not all peaks are covered by the autoregressive test, the inital ones are triggered and sometimes more. Unfortunately, although this design could be interesting in terms of stability, in terms of logic usage, it fails to meet any feasible constraints regarding available logic on low-level FPGA's while remaining at this stability level. The total sizes of the Verilog file sizes of the perceptron-based LUTs are shown in Table 7.16.

NN Layers:	**Input (8)**	**Hidden (6+4)**	**Output (1)**	**Total (19)**
Verilog file size before compression [MB]	6,280	26.0	2.9	6,308.9
Verilog file size after compression [MB]	123.0	9.1	2.3	134.4
Reduction [%]	98.0	64.9	20.7	97.9

Table 7.16: Generated Verilog file sizes for the Design 2 voltage-membrane Neural-Network LUT before and after compression

NN Layers:	**Input (8)**	**Hidden (6+4)**	**Output (1)**	**Total (19)**
LUT size before compression [bit]	1,073,741,824	5,242,880	786,432	1,079,771,136
LUT size after compression [bit]	50,154,895	1,997,504	475,690	52,628,089
Reduction [%]	95,3	61,9	39,5	95,1

Table 7.17: Total LUT size in bits of the Design 2 voltage-membrane Neural-Network LUT before and after compression.

Metrics Evaluation

The reduction in internal precision reduces the size of the perceptron LUTs greatly, since LUT size scales exponentially with input bit width. However, this does introduce a risk of approximation error, potentially reducing the stability of the autoregressive neuron simulation. Hence, the relative reduction in stability levels compared with the previous optimisation in Figure 4.14 is justified, as is the second significant reduction in logic usage.

The optimisation represents the final hardware-accuracy trade-off. A successful configuration should provide a substantial reduction in FPGA resources while preserving sufficient biological behaviour for integration into the neuronal wrapper for scaling. This was best achieved with Design 4 [HYBRID]. However, it would also be interesting to test Design 3 [IONIC-SUM], while Design 2 [SPLIT] would have to be left behind.

7.2.5. Overview Neuron-Core Results

The following matrix in Table 7.18 summarises the explored design space. The colour gradient from red to green illustrates the transition from physically infeasible or numerically unstable implementations towards practical hardware solutions. Each optimisation stage builds on the observations from the previous stage, ultimately yielding a small set of candidate architectures that satisfy the neuron-core requirements: Long-term Stability, Efficient Logic Utilisation, and Reduced DSP Utilisation.

As shown in Table 7.18, the optimisation process progressively eliminates architectures that either remain physically infeasible or fail to maintain stable recursive behaviour. The final results indicate that the HYBRID architecture, combined with delta-prediction Neural Network-generated LUTs and reduced internal bit resolutions, provides the most favourable balance between stability, logic utilisation and DSP usage. At the same time, the IONIC-SUM architecture remains an interesting alternative due to its modular structure and potential to extend to more complex ion-channel models while remaining DSP-less and moderately stable.

	MONOLITH	SPLIT	IONIC-SUM	HYBRID
Baseline (0)	LUT size infeasible	Voltage LUT size infeasible	Only Sodium-LUT infeasible	Feasible, but still huge logic usage
Compressed (1)	-	-	Huge reduction Sodium LUT still large	Huge reduction Still too logic-heavy
Delta Prediction (2)	-	-	Stability better Sodium LUT too large	Stability better Too logic-heavy
NeuraLUT (Control*)	Autoregressive system unstable One-step prediction moderate	-	-	Autoregressive system unstable
NeuraLUT (3)	Autoregressive system unstable One-step prediction accurate	-	-	Single & burst peaks stable Too logic-heavy
Bit Resolution Reduction (4)	-	Single-peak stable Long-term unstable V_m LUT too much logic	Single-peak stable Long-term unstable **13k LUTs / 0 DSP**	Stable & accurate **6.3k LUTs / 4 DSP**

*Control experiment of the NeuraLUT implementation without the Delta-prediction optimisation

Least optimal — Most optimal

Table 7.18: Design-space exploration matrix summarising the results from the optimisation trajectory (1 to 4) on specific design architectures (MONOLITH to HYBRID). The colour scale encodes a qualitative progression from infeasible configurations (red) to near-optimal hardware trade-offs (green) across stability, LUT usage, and DSP cost.

7.3. Neuronal Network

From the design and optimisation trajectory visualised in Table 7.18, Design 4 showed the best trade-off with respect to the Neuron-Core requirements. Hence, this design was used for the first proof-of-concept implementation, in which the core was scaled up to a network of such neurons, pipelined to meet the real-time and high-throughput constraints of the neuron-core wrapper. These results, along with the specific final configuration of synapse weight ratios and other network settings, are presented in Section 7.3.1. Hereafter, even though the stability metric was not as high as preferred, Design 3 was also tested in this neuronal network implementation. Since the resource usage of this design was sufficiently low, the contribution of this experiment is to observe the impact of a less-stable core design on the neuron-core wrapper implementation within a full network of neurons. These results are presented in Section 7.3.2.

7.3.1. Network with Design 4 [HYBRID]

The proof-of-concept consists of 1,250 neurons, with each neuron updated at a simulation timestep of 0.01 ms while operating at a 125 MHz FPGA clock frequency. The network configuration uses an 80%-20% ratio of excitatory and inhibitory neurons, with synaptic weights of 0.06 and -0.36, respectively, yielding an absolute ratio of 1:6. These parameters were experimentally selected through iterative evaluation of different excitation-inhibition ratios to obtain a balanced network behaviour in which neurons receive continuous stochastic synaptic input rather than pre-defined spike timings from a testbench. Additionally, a stochastic external stimulus is applied to initiate and maintain network activity. At every timestep, randomly selected neurons receive a random low-magnitude current injection, between -1.5 and +1.5μA, representing background synaptic input from external neuronal populations. This Poisson-based input mechanism is commonly used to reproduce irregular external synaptic drive received by neurons in large connected networks [28]. The combination of excitation and inhibition inputs from connected neurons, as well as stochastic external inputs, provides an opportunity to evaluate whether the implemented network exhibits stable, asynchronous, irregular spiking behaviour similar to established spiking neuronal network models. Figure 7.17 presents the resulting network activity.

The top plot shows the raster representation of all neuron spikes over biological time. Each black dot represents an individual spike event, where the vertical axis identifies the neuron index and the horizontal axis shows the time occurrence. The highlighted red markers correspond to neuron 0 and are used to see the relation between the global network activity and the detailed voltage trace of that neuron at the bottom of the figure.

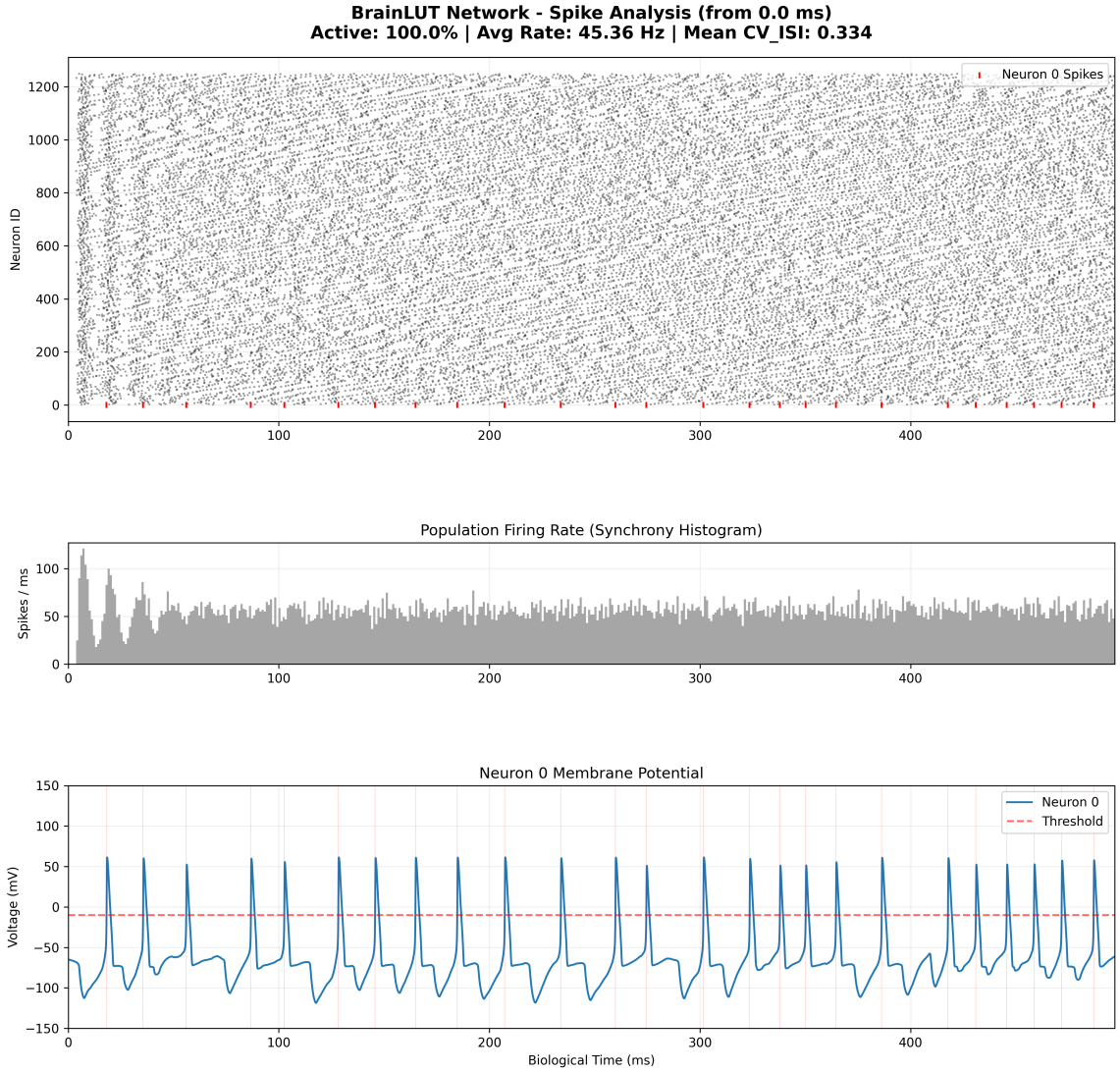


Figure 7.17: Neuronal network simulation using the Design 4 [HYBRID] core for 1,250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.06 and -0.36. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference neuron (bottom).

The middle plot presents the population firing rate using a histogram. This metric is used to evaluate whether the network activity is distributed over time or dominated by a number of separate, synchronised firing events. For a biologically plausible asynchronous network state, the firing activity is expected to remain distributed rather than concentrated into isolated bursts. Therefore, a relatively uniform spike distribution indicates that the network maintains ongoing activity rather than collapsing into excessive synchronisation. Conversely, a synchronous firing regime can be induced by modifying the excitatory-inhibition balance, for example, by increasing the relative contribution of excitatory synaptic weights. This strengthens the propagation of spikes between neurons and can result in large groups of neurons firing within the same time interval, producing burst-like behaviour.

The bottom plot shows the membrane potential of neuron 0 during the same simulation interval. The voltage trace demonstrates the expected interaction between excitatory and inhibitory synaptic inputs. Excitatory inputs increase the membrane potential towards the firing threshold, eventually triggering an action potential, whereas inhibitory inputs counteract this depolarisation and reduce the likelihood of an immediate spike. This demonstrates that the integrated network is not only producing spike activity, but that the underlying Hodgkin-Huxley dynamics continue to respond to synaptic interactions.

An additional observation is that several periods, likely driven by inhibitory input, force the membrane potential towards relatively low values, reaching below -100 mV. This behaviour is likely due to the selected excitation-inhibition weight balance, in which strong inhibitory inputs can temporarily dominate the intrinsic HH dynamics. Despite this, the voltage returns towards the resting state after these events, indicating that the internal dynamics remain stable and can recover from strong inhibitory drives. This suggests that further tuning of synaptic weights could be done to adjust the balance between inhibition strength and desired firing behaviour.

Based on the resulting activity pattern, the network exhibits irregular, asynchronous spiking behaviour, as desired for this first proof-of-concept implementation. To quantify the behaviour, the mean firing rate and coefficient of variation of the inter-spike interval (CV_{ISI}) are reported at the top of the figure. The average firing rate is the mean number of spikes generated per second across the entire network; in this simulation, it was 45.36 Hz. CV_{ISI} shows irregularity of individual neurons' firing. A value close to one is typically associated with irregular Poisson-like firing, while near-zero values indicate regular synchronous spiking behaviour. This simulation yielded a coefficient of 0.334, which is not very high but sufficient to draw initial evaluations and conclusions from the results.

7.3.2. Network with Design 3 [IONIC-SUM]

Design 3 [IONIC-SUM] was integrated into the same neuronal network implementation to evaluate the impact of the less stable neuron core on a large-scale network. After iterative adjustment of the synaptic weight configuration, a stable operating point was obtained with similar network-level activity. The final excitation and inhibition weights were selected as 0.1 and -0.4, respectively. Although the design previously demonstrated reduced computational complexity, the final optimisations required to make it feasible in hardware resulted in a reduced long-term stability compared with Design 4.

Figure 7.18 shows the resulting network activity. At the network level, the design still produces irregular spiking behaviour, with an active neuron ratio of 87.6%, an average firing rate of 34.50 Hz, and a CV_{ISI} of 0.676, indicating that the neurons which remain active still exhibit some irregular spike timing. However, inspection of individual neuron traces reveals reduced stability compared to Design 4. For example, neuron 0 remains inactive throughout the simulation and exhibits oscillatory behaviour below the resting potential without reaching the firing threshold. The corresponding voltage trace is provided in Figure B.19 in Appendix Section B.5.1. This type of behaviour explains the reduced active neuron ratio, as neurons that remain below the firing threshold do not contribute to the network activity. Consequently, Design 3 achieves an active neuron ratio of 87.6%, compared to 100.0% for Design 4.

On top of this, active neurons show altered spike shapes, in which an action potential during its repolarisation is sometimes interrupted by a small secondary peak. This indicates either deviations in the reconstructed neuron dynamics or a non-optimal synaptic weight configuration.

These observations demonstrate that although Design 3 can produce meaningful network-level activity, reduced neuron-core stability affects individual neuron behaviour in a multi-neuron environment. Further investigation would be required to identify whether these instabilities originate from the neuron-core approximation, reduced resolution or the selected network parameters. Therefore, this architecture represents a feasible low-resource alternative, but with the caveat of reduced stability in long-term simulations.

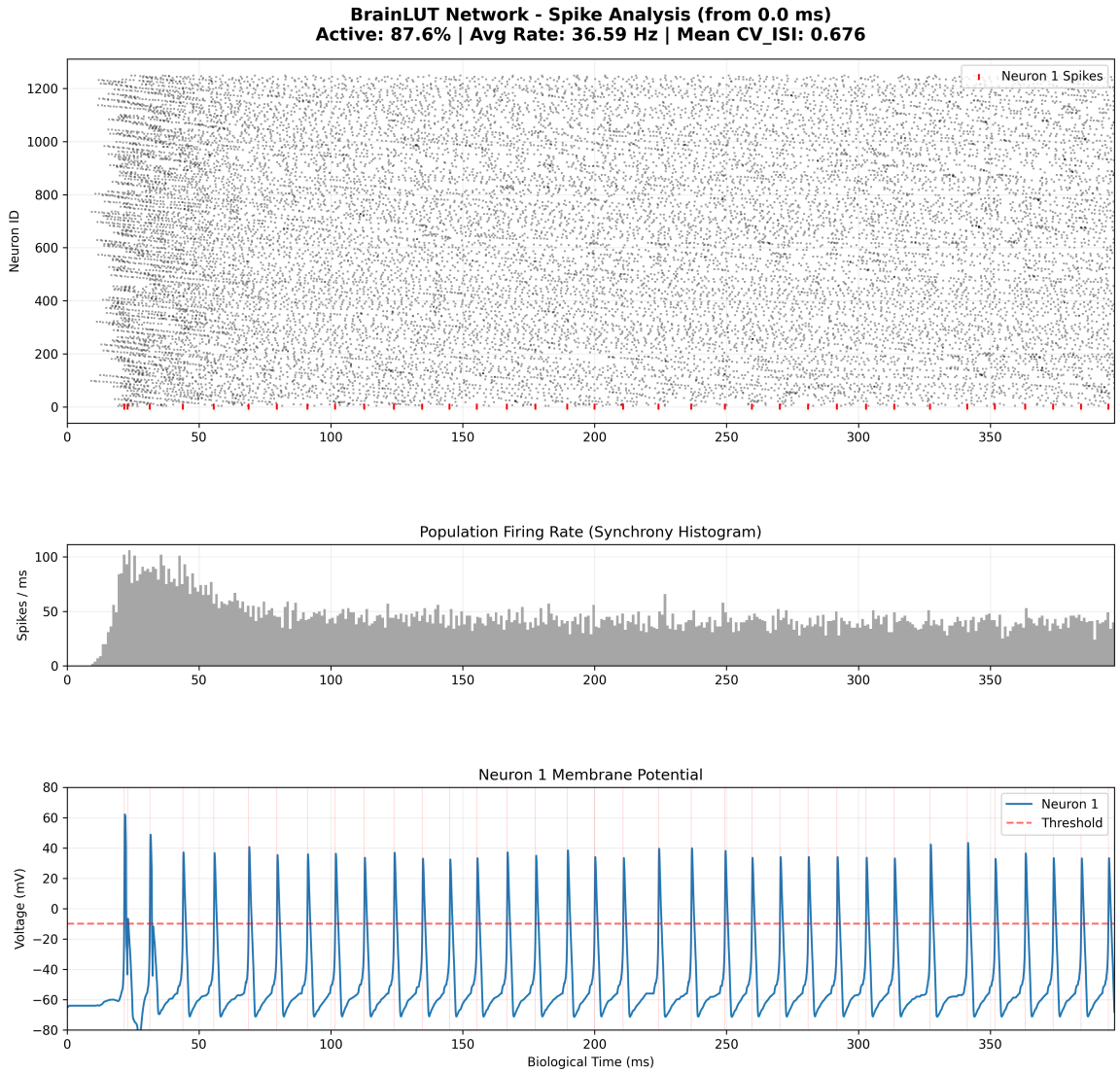


Figure 7.18: Neuronal network simulation using the Design 3 [IONIC-SUM] core for 1,250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.1 to -0.4. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference neuron (bottom).

7.3.3. Timing Performance

Timing analysis confirms that both final implementations satisfy the timing constraints on the Artix-7 device. For the Artix-7 implementation, a parametric sweep of the clock constraint was performed, starting at 10 ns and gradually reducing the period until timing closure could no longer be maintained. The lowest clock period that still resulted in positive slack for both designs was 8.0 ns, corresponding to a maximum operating frequency of 125 MHz. As summarised in Table 7.19, Design 3 achieves a worst-case slack of 0.73 ns with a power requirement of 0.841 W, while Design 4 achieves a worst-case slack of 0.81 ns with a power requirement of 0.495 W.

	Design 3 [IONIC-SUM]	Design 4 [HYBRID]
Target clock frequency	125 MHz	125 MHz
Clock period	8.0 ns	8.0 ns
Worst-case data path delay	7.171 ns	6.577 ns
Worst-case slack	0.734 ns	0.813 ns
Timing constraint met	Yes	Yes
Total On-Chip Power (W)	0.841	0.495

Table 7.19: Place-and-route timing and power results for the final neuronal network implementations Design 3 [IONIC-SUM] and Design 4 [HYBRID] on an Artix-7 XC7A200T FPGA device

This indicates that both designs complete their longest combinational path within the available 8 ns clock period. Therefore, the final chosen clock frequency of 125 MHz is feasible for real-time operation at a simulation timestep of 0.01 ms.

To further evaluate the portability of the implementations across FPGA platforms, the same implementations were additionally synthesised for an Artix UltraScale+ XCAU10P device. This device represents an alternative low-resource FPGA target to the XC7A200T used throughout this work and is based on a more recent 16 nm FinFET process node, compared to the 28 nm technology of the Artix-7 family. A similar constraint sweep was performed on this target, starting from the Artix-7 derived timing and reducing the clock period until the minimum feasible value with positive slack was reached. In this case, both implementations remained timing-closed at a 5.0 ns clock period, corresponding to a maximum operating frequency of 200 MHz, as shown in Table 7.20, without requiring any RTL modifications.

	Design 3 [IONIC-SUM]	Design 4 [HYBRID]
Target clock frequency	200 MHz	200 MHz
Clock period	5.0 ns	5.0 ns
Worst-case data path delay	3.916 ns	4.152 ns
Worst-case slack	0.982 ns	0.720 ns
Timing constraint met	Yes	Yes
Total On-Chip Power (W)	1.002	0.611

Table 7.20: Place-and-route timing and power results for the final neuronal network implementations Design 3 [IONIC-SUM] and Design 4 [HYBRID] on an Artix UltraScale+ XCAU10P FPGA device

The resource utilisation between the two target devices remained within the same range, with only minor variations in the number of utilised logic resources (on the order of tens of LUTs). Therefore, the improved timing performance observed on the Artix UltraScale+ device is not due to an adapted implementation but rather to the characteristics of the target FPGA technology.

Although the XC7A200T is used as the primary evaluation platform for fair comparison with related work, the additional synthesis on the Artix UltraScale+ XCAU10P device indicates that the proposed architecture scales to higher operating frequencies without modification. This suggests that, under higher-frequency implementations, the effective neuron throughput per pipeline would increase proportionally, further improving the achievable neuron density beyond the 1,250-neuron baseline reported for the primary device.

7.3.4. Network Evaluation

The neuronal-network evaluation demonstrated that the proposed neuron-core wrapper can successfully implement the optimised Hodgkin-Huxley core into a scalable multi-neuron architecture. Design 4 [HYBRID] achieved the most favourable balance among neuron core stability, efficient logic usage, and reduced DSP utilisation. At the same time, Design 3 [IONIC-SUM] demonstrated that further resource

reduction is possible at the cost of long-term stability.

Furthermore, the results highlight the importance of the target FPGA technology when evaluating achievable performance. As introduced at the beginning of this chapter, the Artix UltraScale+ XCAU10P device was included as an additional target to assess the impact of device selection. The achieved increase from 125 MHz to 200 MHz demonstrates that the proposed architecture can benefit significantly from newer FPGA architectures without requiring design modifications. However, to maintain a consistent basis for comparison when evaluating against the defined requirements and related work, the XC7A200T remains the primary evaluation platform throughout this thesis.

Requirements

With respect to the neuron-core wrapper requirements defined in Section 3.1.2, Design 4 satisfies the memory-utilisation requirement, using only 7 BRAM-36 blocks and 8 BRAM-18 blocks for the complete system. This resource usage remains sufficiently low to allow scaling towards larger networks. Although Design 3 requires slightly more resources (8 BRAM3-36 and 13 BRAM-16 blocks), both remain within practical limits for any low or mid-range FPGA device.

The real-time operation requirement is met by the pipelined neuron-core architecture. With an FPGA clock period of 8 ns, corresponding to an operational frequency of 125 MHz, the implemented system can update the complete network within the simulation timestep of 0.01 ms. Both designs had positive slack, confirming that they meet the requirement without violating timing constraints.

The final achieved neuron throughput is determined by the maximum number of neurons the system can process per simulation step. In the current proof-of-concept phase, one pipeline within the neuron-core wrapper supports 1,250 neurons in real-time. However, the architecture was designed to allow duplication of these pipelines if additional FPGA resources are available. As a rough estimate, assuming the cumulative amount of resources used by the neuron-core wrapper presented in Table 7.11, approximately three independent neuron-core pipelines could be implemented on the smallest, low-end device of the Artix-7 family, the XC7A35T. This would, in turn, mean that the FPGA could theoretically simulate 3750 neurons in real time. Naturally, if a larger device is chosen, such as the mid-range XC7200T with 133K LUTs at its disposal, with a suitable extended wrapper, it could simulate roughly 21 pipelines to facilitate more than 25 thousand neurons.

Overall, Design 4 is selected as the final neuron-core configuration and benchmarked for performance comparison, as it provides the most stable behaviour while maintaining sufficiently low resource utilisation. The following chapter evaluates this final architecture against existing FPGA-based Hodgkin-Huxley implementations from Chapter 2 to position the proposed approach within the current state of the art. Throughout the remainder of this thesis, the proposed architecture is referred to as BrainLUT to provide a consistent naming convention for the complete system.

One final discussion point is that the neuron core was optimised as an independent hardware module throughout this work before being scaled to a multi-neuron network. In principle, future optimisation could consider the complete neuronal network as the design target from the outset. The current optimisation primarily focuses on the computational efficiency and numerical behaviour of a single neuron, whereas a large-scale network also introduces communication, routing, and memory-access overheads that affect overall implementation efficiency. Consequently, neuron architectures with different resource or latency trade-offs could become preferable when evaluated at the network level rather than in isolation.

7.4. Performance Comparison

To place the results in the context of state-of-the-art designs, the final implementation is compared with a selection of FPGA-based Hodgkin-Huxley neuron implementations reported in the literature. These works were introduced earlier in Chapter 2. Since the reported designs target different FPGA families, neuron models, and simulation time-step resolutions, direct comparison of absolute neuron counts is not meaningful. Instead, the comparison focuses primarily on hardware-independent implementation metrics, including LUTs and DSPs per neuron, along with operating frequency and power consumption, where available. These normalised metrics provide a fairer basis for evaluating architectural efficiency across different FPGA platforms.

Under a real-time simulation constraint of $\Delta t = 0.01$ ms, the proposed BrainLUT architecture supports 1250 neurons on an Artix-7 XC7A200T device while using 6307 LUTs and only four DSP blocks, corresponding to 5.05 LUTs and 0.0032 DSPs per neuron.

Work	Process Node (nm)	Device	#LUT/N	#DSP/N	Op.Freq. (MHz)	Power/N (mW)	GFLOPS/N	Model	Δt (μs)	Neurons @RT
BrainLUT (This Work)	28	Artix 7 XC7A200T	**5.05**	**0.0032**	**125**	**0.4**	**0.0012***	**HH**	**10.0**	**1,250**
ReLANCE [48]	28	Virtex 7 VC707	1,853.1	0	**198.4**	411.0	12.69*	(CORDIC) HH	–	64
BicemuS [49]	16	Zynq UltraScale+ KR260	28.6	0.47	–	6.3	–	**Pospischil** HH	31.3	**1,024**
Beaubois et al. [10]	16	Zynq UltraScale+ KR260	42,543	519	–	350	–	**Pospischil** HH	100.0	16
Zhang et al. [50]	90	Spartan-3A XC3SD1800A	1,298.0	0.80	100.0	–	–	HH	100.0	5
Yi et al. [51]	28	Zynq-7000	763.7	4.06	10.0	–	–	HH	**10.0**	11*
Akbarzadeh-Sherbaf et al. [12]	28	Artix-7 XC7A200T	7.2	0.068	71.4	–	–	HH	**7.8**	**4,096**
Khoyratee et al. [36]	28	Kintex-7 XC7K32T	11.2	0.056	100.0	–	–	HH	32.0	**15,000**
Bonabi et al. [52]	28	Virtex 7 XQ7K410T	573.5	7.40	63.4	16.7	–	HH	100.0	150
flexHH [53]	28	Stratix V 5SGSD8	–	–	**180.0**	3.9–4.5	11.22	HH	60.0	**10,368**
flexHH (full connectivity) [53]	28	Stratix V 5SGSD8	–	–	**180.0**	3.9–4.5	11.22	Extended HH	**10.0**	166
Zjajo et al. [54]	28	Virtex 7 XC7VX550	262.5	0.84	100.0	–	2.13	Extended HH	50.0	**1,188**
Smaragdos et al. [55]	40	Virtex 6 SX475T	577.2	1.80	150.0	–	0.0748	Extended HH	50.0	330
Pourhaj et al. [56]	65	Virtex 5 V5-110t	42.4	–	**173.8**	–	0.185	Extended HH	10.0	**1,024**
Luo et al. [57]	28	Virtex 7 XC7VX485T	**32,142**	64.3	2.86	–	763.7	Extended HH	50.0	500

**Estimated using a computational equivalence of 1200 FLOPs per Hodgkin–Huxley neuron model per millisecond simulation step [22]*

Table 7.21: Performance comparison of the proposed neuronal network design with the Design 4 [HYBRID] neuron core with related work from Table 7.21, using a selected set of comparable metrics

Figure 7.19 visualises the normalised logic and DSP requirements of the surveyed implementations, illustrating that BrainLUT achieves the lowest LUT utilisation per neuron among the designs compared while also requiring the lowest DSP usage per neuron. Although several published implementations realise more complex neuron variants, such as extended Hodgkin-Huxley formulations or multicompartment models, which inherently require additional hardware resources, the comparison demonstrates that BrainLUT achieves an exceptionally low hardware cost for the implemented neuron model.

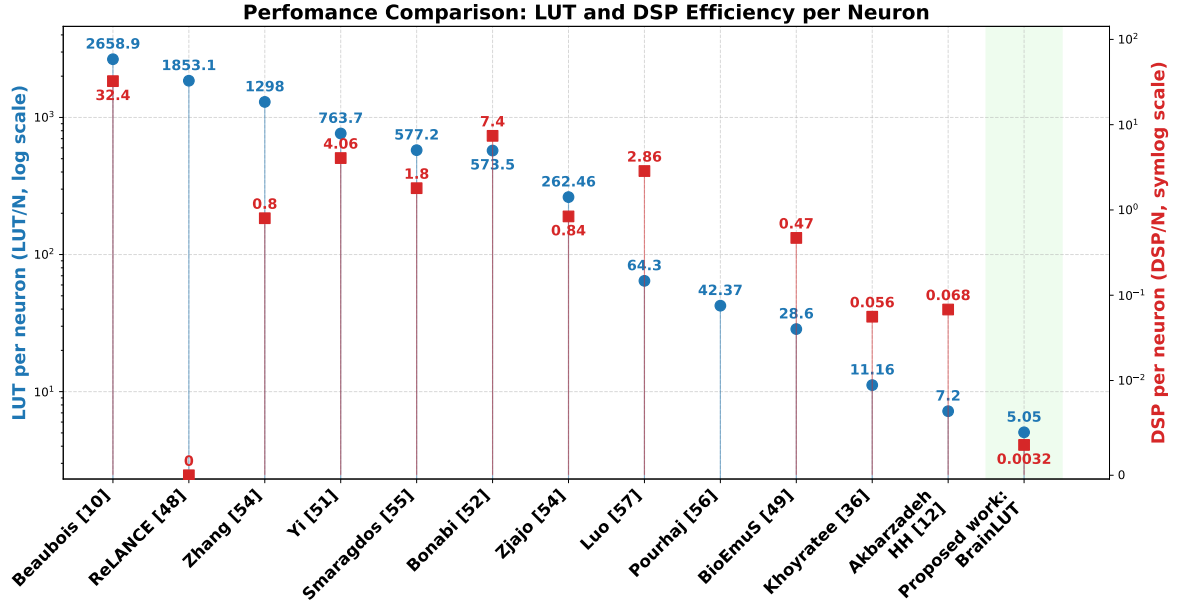


Figure 7.19: Visual comparison of the proposed work and related works based on the normalised Logic and DSP count per neuron

Compared to prior FPGA-based HH systems, including flexHH [53], Bonabi et al. [52], Zjajo et al. [54], and Smaragdos et al. [55], some implementations report higher absolute neuron counts. However, these figures are largely influenced by the available FPGA resources and the degree of architectural

replication rather than the efficiency of an individual neuron implementation. When evaluated using the normalised per-neuron metrics, BrainLUT requires substantially fewer logic and DSP resources than the surveyed implementations, demonstrating a highly efficient hardware realisation of the Hodgkin-Huxley model.

It should be noted that the reported capacity of 1250 real-time neurons reflects the implementation of the architecture on the selected Artix-7 evaluation platform rather than an architectural limit. Mapping the same design onto larger or more modern FPGA devices would enable additional pipeline replication and, correspondingly, higher neuron counts, subject to routing, memory bandwidth, and available hardware resources. Consequently, the normalised implementation metrics presented above provide a more representative measure of the architectural efficiency than the absolute neuron capacity alone.

The primary advantage lies in the drastic reduction of DSP usage, achieved through LUT-based approximation of the HH dynamics. This reduces per-neuron arithmetic cost to near-minimal levels while maintaining deterministic real-time execution. In contrast, many prior FPGA implementations still rely on per-neuron arithmetic pipelines or DSP-heavy evaluation of nonlinear functions.

Although some existing architectures achieve higher absolute neuron counts, they typically require substantially larger FPGA resources or higher per-neuron arithmetic complexity, which in turn increases overall power consumption due to their reliance on arithmetic-intensive datapaths. When normalised by resource usage, the proposed design achieves a favourable balance between computational efficiency and scalability, with improved power efficiency emerging as a secondary consequence of the reduced computational complexity, as seen in Table 7.19.

Finally, the proposed architecture maintains this resource efficiency while operating at a simulation time step of 0.01 ms, which is finer than that used in many of the surveyed implementations. Consequently, the presented results demonstrate that the proposed LUT-based formulation enables highly efficient real-time Hodgkin-Huxley neuron simulation without sacrificing temporal resolution.

8

Conclusions

This thesis investigated how Neural Network-based LUT representations can increase the achievable density of Hodgkin–Huxley neuron simulations on FPGA architectures under a fixed resource budget while maintaining physiologically realistic multi-neuron dynamics.

The proposed approach demonstrated that Neural Network-based LUTs can successfully represent the nonlinear dynamics of the Hodgkin-Huxley model and transform these into FPGA-efficient logic structures. Across the developed optimisation pipeline—including architectural decomposition, LUT compression, delta prediction, NeuraLUT conversion, and layer-specific resolution reduction—the computational cost of the neuron model was reduced while preserving stable single-neuron and network-level behaviour, enabling higher-density FPGA deployment.

Evaluation of the design space shows that a fully DSP-less implementation is feasible, confirming the representational flexibility of LUT-based neural approximations in removing dependence on dedicated arithmetic hardware. However, in the current Neural Network configuration, removing all arithmetic support increases sensitivity to quantisation effects, which degrades long-term stability in both isolated neurons and network-scale simulations. In contrast, a low-DSP hybrid configuration mitigates these effects and achieves a more balanced trade-off between stability, biological accuracy, and neuron density, indicating that selective reintroduction of arithmetic operations can improve robustness without significantly compromising resource efficiency.

The developed neuronal network provides a successful first proof of concept, demonstrating that a single FPGA can execute a real-time Hodgkin-Huxley-based neuronal network using Neural Network-derived LUTs in combination with LUT compression. The network successfully integrates a time-multiplexed neuron core pipeline, synaptic connectivity, spike routing, and scheduling, demonstrating that the proposed architecture can be integrated into network-level brain simulation.

To formalise the architectural trade-offs, the explored designs are summarised in Table 8.1. The comparison reveals a progression from highly simplified but non-scalable configurations (Design 1) toward increasingly modular and scalable architectures (Designs 3 and 4). Design 1 (MONOLITH) prioritises simplicity and DSP efficiency at the expense of modularity and scalability, while Design 2 (SPLIT) introduces partial decomposition without sufficient gains in resource flexibility. Design 3 (IONIC-SUM) provides a balanced intermediate design and serves as the primary reference point for optimisation and comparison throughout the evaluation. The thesis then ultimately converges on Design 4 (HYBRID) as the final architecture due to its superior balance between biological fidelity, FPGA resource utilisation, and scalability. Its improved separation of conductance and gating pathways enhances modularity, stabilises NeuraLUT conversion, and reduces quantisation-induced drift in long-term simulations. While structurally more complex than Design 3, it yields improved robustness without compromising efficiency. This convergence is consistent with the observed optimisation results, where LUT compression, delta-prediction reformatting, and resolution scaling improved all designs but did not eliminate the stability advantage of the hybrid architecture.

Property	Design 1	Design 2	Design 3	Design 4
Simplicity	★★★★★	★★★★⋆	★★★⋆⋆	★★⋆⋆⋆
Modularity	★⋆⋆⋆⋆	★★⋆⋆⋆	★★★★⋆	★★★★★
Efficient Logic utilisation	★⋆⋆⋆⋆	★★⋆⋆⋆	★★★★⋆	★★★★★
Reduced DSP utilisation	★★★★★	★★★★★	★★★★★	★★⋆⋆⋆
Stable with NeuraLUT	★⋆⋆⋆⋆	★★★⋆⋆	★★★⋆⋆	★★★★★
Stable in Neuronal Network	⋆⋆⋆⋆⋆	⋆⋆⋆⋆⋆	★★★⋆⋆	★★★★★
Scalability	★⋆⋆⋆⋆	★★⋆⋆⋆	★★★★⋆	★★★★★

Table 8.1: Qualitative final design comparison after optimisations across architectural criteria using star ratings

When placed in the context of state-of-the-art FPGA-based Hodgkin–Huxley implementations, the proposed design demonstrates substantially improved resource efficiency, particularly through the near-elimination of DSP utilisation and substantial reductions in normalised logic usage per neuron. In addition, the architecture achieves lower per-neuron power consumption as an emergent consequence of reduced arithmetic intensity. Beyond resource efficiency, the proposed approach exhibits favourable scaling with respect to neuron density under fixed FPGA budgets. While absolute neuron counts depend on the chosen device and time-multiplexing configuration, the architecture supports further linear scaling through increased clock frequency and pipeline replication. This positions the design as highly competitive in both efficiency and scalability compared to existing FPGA-based implementations.

However, the presented optimisation strategies are dependent on the targeted low-resource FPGA architectures. On future FPGA platforms with increased memory bandwidth and capacity, such as devices equipped with High Bandwidth Memory (HBM), the trade-off space may shift. Since memory capacity and bandwidth become less restrictive, maintaining higher bit resolutions and larger LUT representations may become feasible, reducing quantisation errors and potentially improving biological accuracy without significantly affecting scalability.

8.1. Main Contributions

The main contributions of this thesis are as follows:

- A pipelined Hodgkin-Huxley neuron core architecture based on shared LUT resources for efficient representation of channel-dependent nonlinear dynamics, enabling significantly increased neuron density under FPGA resource constraints.
- An adapted version of the ReducedLUT algorithm, enabling compression of larger LUT representations while maintaining synthesizable FPGA implementations.
- An adapted NeuraLUT-based conversion flow for transforming quantisation-aware trained neural networks into hardware LUT representations suitable for higher-resolution function approximation and scalable FPGA deployment.
- A flexible methodology for modifying learned channel dynamics and extending the architecture towards additional ionic channels not included in the classical Hodgkin-Huxley model.
- A first proof-of-concept implementation of a real-time Hodgkin-Huxley neuronal network architecture on a resource-constrained FPGA, demonstrating scalable neuron-density growth through time-multiplexed execution and efficient resource utilisation.

8.2. Future Work

Future research can further improve both the neuron core and the network architecture. The possible future directions are presented in this section.

8.2.1. Neuron-Core Suggestions

Design 2 and Design 3 can be further optimised by training neural networks to learn the separate state updates simultaneously. Since separate training of individual states was successful on individual tests but performed worse when integrated into a combined system, future research could focus on implementing this combination of separate networks during training rather than after. By allowing each

network's predictions to be used not only by itself for the next step but also by other networks that share the same variable state, the neural networks rely fully on predicted states rather than on a single predicted input and a single true input, usually the voltage state. This way, individual networks may benefit from correlated errors, improving overall accuracy.

Additionally, exploring a wider range of possible neural network hyperparameters, MLP configurations and depths, and ranges of quantisation resolution could provide a better understanding of the optimal accuracy versus resource efficiency trade-off. Future work could perform a systematic hyperparameter search to quantify this trade-off.

Moreover, exploring the trade-off between the latency of a single neuron core update and pipeline throughput could benefit from deeper pipelines by splitting the longest combinational paths within the system or by adding more hidden layers in the Neural Network-derived LUTs.

Finally, in terms of biological plausibility, additional ionic channels could be learned by the adapted NeuraLUT system and integrated into the architecture to affect the neuron dynamics in more detail. Even though for Design 2 [SPLIT], it would mean retraining the coarse-grained voltage LUT, for Design 3 [IONIC-SUM] and Design 4 [HYBRID], only adding the new channel LUTs to the final summation is required.

8.2.2. Neuronal-Network Future Work Directions

Regarding the neuronal network, future work should focus on upscaling the neuronal wrapper architecture by improving memory utilisation, for example, through more efficient BRAM allocation and reduced state-storage overhead in accumulators. Once these constraints are mitigated, the architecture can be extended by replicating the neuron-core pipeline and spike-processing system (FSM and routing logic), enabling higher levels of parallelism and increased neuron density, potentially reaching tens of thousands of neurons depending on device capacity.

An additional important direction is the exploration of alternative FPGA platforms and technology nodes. The results in this thesis indicate that achievable neuron density is strongly influenced not only by resource utilisation, but also by timing closure and maximum clock frequency. Newer FPGA families with improved routing architectures, higher logic interconnect efficiency, and increased memory bandwidth can enable higher operating frequencies without modifying the RTL design. Since increasing the clock frequency directly increases neuron update throughput in a time-multiplexed pipeline architecture, the maximum achievable neuron density scales linearly under fixed resource budgets. Future work could therefore systematically evaluate the same architecture across multiple FPGA generations to explore this frequency–density optimum.

Finally, to increase biological plausibility, future research should investigate extending the architecture with multicompartment Hodgkin-Huxley models, in which the inter-neuron connecting axon membrane equations are explicitly implemented as hardware-efficient representations. This spatial representation would enable accurate synaptic propagation delays and other local electrical dynamics while maintaining the hardware-efficient principles developed in this thesis.

Overall, the work demonstrates that LUT-based neural approximations can serve as a scalable design foundation for high-density FPGA-based neural simulation, in which both resource efficiency and clock-frequency-driven throughput jointly determine the achievable neuron density.

References

[1] World Health Organization, “Neurological disorders affect millions globally: Who report,” World Health Organization, Tech. Rep., Feb. 2007, Accessed: 2026-06-19. [Online]. Available: https://www.who.int/news/item/27-02-2007-neurological-disorders-affect-millions-globally-who-report.

[2] R. A. Barker, M. Götz, and M. Parmar, “New approaches for brain repair-from rescue to reprogramming,” *Nature*, vol. 557, no. 7705, pp. 329–334, May 2018, ISSN: 1476-4687. DOI: 10.1038/s41586-018-0087-1. [Online]. Available: https://doi.org/10.1038/s41586-018-0087-1.

[3] M. Chiappalone and T. Levi, “Advancing neuroengineering with neuromorphic twins,” *Nature Communications*, vol. 17, no. 1, 2026. DOI: 10.1038/s41467-026-68923-1.

[4] H. Keren, J. Partzsch, S. Marom, and C. G. Mayr, “A biohybrid setup for coupling biological and neuromorphic neural networks,” *Frontiers in Neuroscience*, vol. Volume 13 - 2019, 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00432. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00432.

[5] C. D. Schuman et al., *A survey of neuromorphic computing and neural networks in hardware*, 2017. arXiv: 1705.06963 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1705.06963.

[6] M. Ward and O. Rhodes, “Beyond lif neurons on neuromorphic hardware,” *Frontiers in Neuroscience*, vol. Volume 16 - 2022, 2022, ISSN: 1662-453X. DOI: 10.3389/fnins.2022.881598. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.881598.

[7] M. D. Florio, A. Kahana, and G. E. Karniadakis, “Analysis of biologically plausible neuron models for regression with spiking neural networks,” *ArXiv*, vol. abs/2401.00369, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:266693597.

[8] A. L. Hodgkin and A. F. Huxley, “Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo,” *The Journal of Physiology*, vol. 116, no. 4, pp. 449–472, 1952. DOI: https://doi.org/10.1113/jphysiol.1952.sp004717. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004717. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1952.sp004717.

[9] R. Miedema and C. Strydis, “Exaflexhh: An exascale-ready, flexible multi-fpga library for biologically plausible brain simulations,” *Frontiers in Neuroinformatics*, vol. 18, 2024. DOI: 10.3389/fninf.2024.1330875.

[10] R. Beaubois, J. Cheslet, Y. Ikeuchi, P. Branchereau, and T. Levi, “Real-time multicompartment hodgkin-huxley neuron emulation on soc fpga,” *Frontiers in Neuroscience*, vol. Volume 18 - 2024, 2024, ISSN: 1662-453X. DOI: 10.3389/fnins.2024.1457774. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2024.1457774.

[11] Y. Liu, X. He, J. Yu, and K. Wang, “Dif-lut: A simple yet scalable approximation for non-linear activation function on fpga,” in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*, 2023, pp. 322–326. DOI: 10.1109/FPL60245.2023.00055.

[12] K. Akbarzadeh-Sherbaf, B. Abdoli, S. Safari, and A.-H. Vahabie, “A scalable fpga architecture for randomly connected networks of hodgkin-huxley neurons,” *Frontiers in Neuroscience*, vol. 12, 2018. DOI: 10.3389/fnins.2018.00698.

[13] A. J. Leigh, M. Heidarpur, and M. Mirhassani, “A resource-efficient and high-accuracy cordic-based digital implementation of the hodgkin–huxley neuron,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 9, pp. 1377–1388, 2023. DOI: 10.1109/tvlsi.2023.3296057.

[14] G. Ghanbarpour, M. Chaudhary, and M. Ghanbarpour, “An optimized digital implementation of the hodgkin–huxley neuron model on fpga: Enhancing computational efficiency for biomedical and neural applications,” *IEEE Transactions on Circuits and Systems I Regular Papers*, vol. PP, pp. 1–12, Aug. 2025. DOI: 10.1109/TCSI.2025.3598849.

[15] E. L. Graas, E. A. Brown, and R. H. Lee, “An fpga-based approach to high-speed simulation of conductance-based neuron models,” *Neuroinformatics*, vol. 2, no. 4, pp. 417–436, 2004. DOI: 10.1385/ni:2:4:417.

[16] P. Pourhaj, D. H.-Y. Teng, K. Wahid, and S.-B. Ko, “A novel scalable parallel architecture for biological neural simulations,” in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 3152–3155. DOI: 10.1109/ISCAS.2010.5537951.

[17] M. Andronic and G. A. Constantinides, “Neuralut: Hiding neural network density in boolean synthesizable functions,” in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024. DOI: 10.1109/FPL64840.2024.00028. [Online]. Available: https://arxiv.org/abs/2403.00849.

[18] O. Cassidy, M. Andronic, S. Coward, and G. A. Constantinides, “Reducedlut: Table decomposition with ”don’t care” conditions,” *arXiv preprint arXiv:2412.18579*, 2024. DOI: 10.48550/arXiv.2412.18579. [Online]. Available: https://arxiv.org/abs/2412.18579.

[19] T. Wang, Y. Wang, J. Shen, L. Wang, and L. Cao, “Predicting spike features of hodgkin-huxley-type neurons with simple artificial neural network,” *Frontiers in Computational Neuroscience*, vol. Volume 15 - 2021, 2022, ISSN: 1662-5188. DOI: 10.3389/fncom.2021.800875. [Online]. Available: https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2021.800875.

[20] Encyclopaedia Britannica, Inc., *Unmyelinated and myelinated axons: Action potential propagation*, Illustration, Figure reprinted from Encyclopaedia Britannica, Inc., 2002.

[21] J. Hutchins and A. Crookston, *Resting membrane potential*, BioRender illustration, Figure created with BioRender.com. Published in *The Resting Membrane Potential*, On Becoming a Neuroscientist, University of Colorado Pressbooks. Licensed under CC BY-NC-ND, 2020. [Online]. Available: https://colorado.pressbooks.pub/neuroscience/chapter/the-resting-membrane-potential/.

[22] E. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004. DOI: 10.1109/TNN.2004.832719.

[23] R. Brette and W. Gerstner, “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity,” *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005. DOI: 10.1152/jn.00686.2005.

[24] K. Tóth and M. Okun, “Seventy years later: The legacy of the hodgkin and huxley model in computational neuroscience,” *The Journal of Physiology*, vol. 601, no. 15, pp. 3009–3010, 2023. DOI: https://doi-org.tudelft.idm.oclc.org/10.1113/JP284606. eprint: https://physoc-onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/pdf/10.1113/JP284606. [Online]. Available: https://physoc-onlinelibrary-wiley-com.tudelft.idm.oclc.org/doi/abs/10.1113/JP284606.

[25] Clinical Gate, *Cardiac electrophysiology*, Online medical textbook chapter, Figure 18-4 accessed online. Accessed: 2026-06-20, 2015. [Online]. Available: https://clinicalgate.com/cardiac-electrophysiology/.

[26] W. Gerstner and W. M. Kistler, *Conductance-based models*, Scholarpedia, Accessed: 2026-06-20, 2008. [Online]. Available: http://www.scholarpedia.org/article/Conductance-based_models.

[27] W. F. Herman, *Introduction to Partial Differential Equations*. LibreTexts, 2022, Figure 10.1.1 accessed online. Accessed: 2026-06-20. [Online]. Available: https://math.libretexts.org/Bookshelves/Differential_Equations/Introduction_to_Partial_Differential_Equations_(Herman)/10%3A_Numerical_Solutions_of_PDEs/10.01%3A_Ordinary_Differential_Equations.

[28] N. Brunel, "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons," *Journal of Computational Neuroscience*, vol. 8, pp. 183–208, 2000.

[29] S. Panagiotou, R. Miedema, H. Sidiropoulos, G. Smaragdos, C. Strydis, and D. J. Soudris, "A novel simulator for extended hodgkin-huxley neural networks," *2020 IEEE 20th International Conference on Bioinformatics and Bioengineering (BIBE)*, pp. 395–402, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:229307443.

[30] E. Yavuz, J. Turner, and T. Nowotny, "Genn: A code generation framework for accelerated brain simulations," *Scientific Reports*, vol. 6, no. 1, 2016. DOI: 10.1038/srep18854.

[31] M. C. W. Engelen, R. Betting, and C. Strydis, "Simhh: A versatile, multi-gpu simulator for extended hodgkin-huxley networks," *IEEE Access*, vol. 13, pp. 46 865–46 880, 2025. DOI: 10.1109/ACCESS.2025.3550444.

[32] J. C. Knight and T. Nowotny, "Gpus outperform current hpc and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model," *Frontiers in Neuroscience*, vol. Volume 12 - 2018, 2018, ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00941. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00941.

[33] M. Isik, *A survey of spiking neural network accelerator on fpga*, 2023. arXiv: 2307.03910 [cs.AR]. [Online]. Available: https://arxiv.org/abs/2307.03910.

[34] J. van der Kleij, "Exploring the use of fpgas in implantable medical devices," MSc Thesis, Delft University of Technology, 2024. [Online]. Available: https://repository.tudelft.nl/file/File_a5cba8da-1879-4465-8a72-c22d0f4d7b70.

[35] I. Kuon, R. Tessier, and J. Rose, "Fpga architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007. DOI: 10.1561/1000000005.

[36] F. Khoyratee, F. Grassia, S. Saïghi, and T. Levi, "Optimized real-time biomimetic neural network on fpga for bio-hybridization," *Frontiers in Neuroscience*, vol. Volume 13 - 2019, 2019, ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00377. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00377.

[37] K. Nakamura et al., "A memory-based programmable logic device using look-up table cascade with synchronous static random access memories," *Japanese Journal of Applied Physics*, vol. 45, no. 4B, pp. 3295–3300, 2006. DOI: 10.1143/JJAP.45.3295.

[38] RapidWright Team, *Fpga architecture basics*, RapidWright Documentation, Figure illustrating LUT architecture accessed online. Accessed: 2026-06-20, 2025. [Online]. Available: https://www.rapidwright.io/docs/FPGA_Architecture.html.

[39] AMD, *7 Series FPGAs Data Sheet: Overview*, https://docs.amd.com/v/u/en-US/ds180_7Series_Overview, Accessed: 2026-06-20, 2020.

[40] AMD, *Virtex UltraScale+ VU19P FPGA Product Table*, https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus-vu19p.html, Accessed: 2026-06-20, 2026.

[41] Intel, *Intel Arria 10 GX 1150 FPGA Specifications*, https://www.intel.com/content/www/us/en/products/sku/210381/intel-arria-10-gx-1150-fpga/specifications.html, Accessed: 2026-06-20, 2026.

[42] Intel, *Intel Stratix 10 SX SoC FPGA Product Table*, https://www.intel.com/content/www/us/en/ark/products/series/141550/intel-stratix-10-sx-soc-fpga.html, Accessed: 2026-06-20, 2026.

[43] H. Patil, *Neural networks explained simply: How machines learn to think*, Medium article, Figure accessed online. Accessed: 2026-06-20, 2026. [Online]. Available: https://medium.com/@patilhemantd73/neural-networks-explained-simply-how-machines-learn-to-think-50cf4b414495.

[44] GeeksforGeeks, *Feedforward neural network*, Online article, Figure illustrating a multilayer perceptron with hidden layer accessed online. Accessed: 2026-06-20, 2025. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/feedforward-neural-network/.

[45] B. Ahn, "Implementation of a 12-million hodgkin-huxley neuron network on a single chip," *CoRR*, vol. abs/2004.13334, 2020. arXiv: 2004.13334. [Online]. Available: https://arxiv.org/abs/2004.13334.

[46] S. Yang et al., "Real-time neuromorphic system for large-scale conductance-based spiking neural networks," *IEEE Transactions on Cybernetics*, vol. 49, no. 7, pp. 2490–2503, 2019. DOI: 10.1109/TCYB.2018.2823730.

[47] S. Haghiri, A. Naderi, B. Ghanbari, and A. Ahmadi, "High speed and low digital resources implementation of hodgkin-huxley neuronal model using base-2 functions," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 1, pp. 275–287, 2021. DOI: 10.1109/tcsi.2020.3026076.

[48] S. Kumar, A. S. Nair, B. Chaudhary, M. Lokhande, and S. K. Vishvakarma, "Relance: A resource-efficient low-latency cortical neural acceleration engine," *arXiv preprint arXiv:2510.17392*, 2026. [Online]. Available: https://arxiv.org/abs/2510.17392.

[49] R. Beaubois et al., "Bicœmus: A new tool for neurological disorders studies through real-time emulation and hybridization using biomimetic spiking neural network," *Nature Communications*, vol. 15, p. 5142, 2024.

[50] Y. Zhang, J. Nunez-Yanez, J. Mcgeehan, E. Regan, and S. Kelly, "A biophysically accurate floating point somatic neuroprocessor," Aug. 2009, pp. 26–31. DOI: 10.1109/FPL.2009.5272558.

[51] Z. Yi, W. Fu, and L. Cao, "Implementation of hodgkin-huxley spiking neuron model using fpga," Oct. 2019, pp. 1535–1539. DOI: 10.1109/ICCT46805.2019.8947169.

[52] S. Yaghini Bonabi, H. Asgharian, S. Safari, and M. Nili Ahmadabadi, "Fpga implementation of a biological neural network based on the hodgkin-huxley neuron model," *Frontiers in Neuroscience*, vol. 8, Nov. 2014. DOI: 10.3389/fnins.2014.00379.

[53] R. Miedema, G. Smaragdos, M. Negrello, Z. Al-Ars, M. Möller, and C. Strydis, "Flexhh: A flexible hardware library for hodgkin-huxley-based neural simulations," *IEEE Access*, vol. 8, pp. 121 905–121 919, 2020. DOI: 10.1109/ACCESS.2020.3007019.

[54] A. Zjajo et al., "A real-time reconfigurable multichip architecture for large-scale biophysically accurate neuron simulation," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 2, pp. 422–434, 2018. DOI: 10.1109/TBCAS.2017.2780287.

[55] G. Smaragdos et al., "Real-time olivary neuron simulations on dataflow computing machines," vol. 8488, Jul. 2014, ISBN: 978-3-319-07517-4. DOI: 10.1007/978-3-319-07518-1_34.

[56] P. Pourhaj and D. H.-Y. Teng, "Fpga based pipelined architecture for action potential simulation in biological neural systems," in *CCECE 2010*, 2010, pp. 1–4. DOI: 10.1109/CCECE.2010.5575160.

[57] J. Luo et al., "Optogenetics in silicon: A neural processor for predicting optically active neural networks," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 1, pp. 15–27, 2017. DOI: 10.1109/TBCAS.2016.2571339.

[58] L. Landsmeer, *Lut_hh: Lookup-table-based hodgkin-huxley neuron model*, https://github.com/llandsmeer/LUT_HH, GitHub repository, accessed June 2026, 2024.

[59] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 25–71, 1990, ISSN: 0092-8240. DOI: https://doi.org/10.1016/S0092-8240(05)80004-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092824005800047.

[60] OpenWorm Project, *Hodgkin-huxley tutorial*, https://github.com/openworm/hodgkin_huxley_tutorial, Accessed: 2026-06-22, 2022.

[61] M. Andronic, J. Li, and G. A. Constantinides, *Polylut: Ultra-low latency polynomial inference with hardware-aware structured pruning*, 2025. arXiv: 2501.08043 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2501.08043.

[62] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[63] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023. [Online]. Available: https://d2l.ai.

[64] Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986. DOI: 10.1109/TC.1986.1676819.

[65] *7 series fpgas memory resources user guide (ug473)*, v1.14, Xilinx, Inc., 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf.

[66] W. Gerstner, Wulfram, Kistler, and W. M., *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, Aug. 2002. DOI: 10.1017/CBO9780511815706.

[67] S. Eissa, S. Stuijk, and H. Corporaal, "Hardware approximation of exponential decay for spiking neural networks," English, in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021*, 3rd IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021 ; Conference date: 03-05-2021 Through 07-05-2021, United States: Institute of Electrical and Electronics Engineers, Jun. 2021. DOI: 10.1109/AICAS51828.2021.9458560.

[68] M. Naumov, L. S. Chien, P. Vandermersch, and U. Kapasi, "Cusparse library," in *GPU Technology Conference (GTC)*, San Jose, CA, USA, Sep. 2010. [Online]. Available: https://www.nvidia.com/content/GTC-2010/pdfs/2070_GTC2010.pdf.

[69] R. M. Wang, G. Cohen, K. M. Stiefel, T. J. Hamilton, J. C. Tapson, and A. van Schaik, "An fpga implementation of a polychronous spiking neural network with delay adaptation," *Frontiers in Neuroscience*, vol. Volume 7 - 2013, 2013, ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00014. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2013.00014.

[70] NeuroComputing Lab, *NeuroComputing Lab (NCL) GitLab Repository*, https://gitlab.com/c7859/neurocomputing-lab, Accessed: 2026-07-06, 2026.

A

ReducedLUT - Adaptation

The full adaptation of the ReducedLUT algorithm, including the reconstruction of the original value, is shown in Figures A.1 to A.3 below. The code behind this process is detailed in the GitHub repository [70].

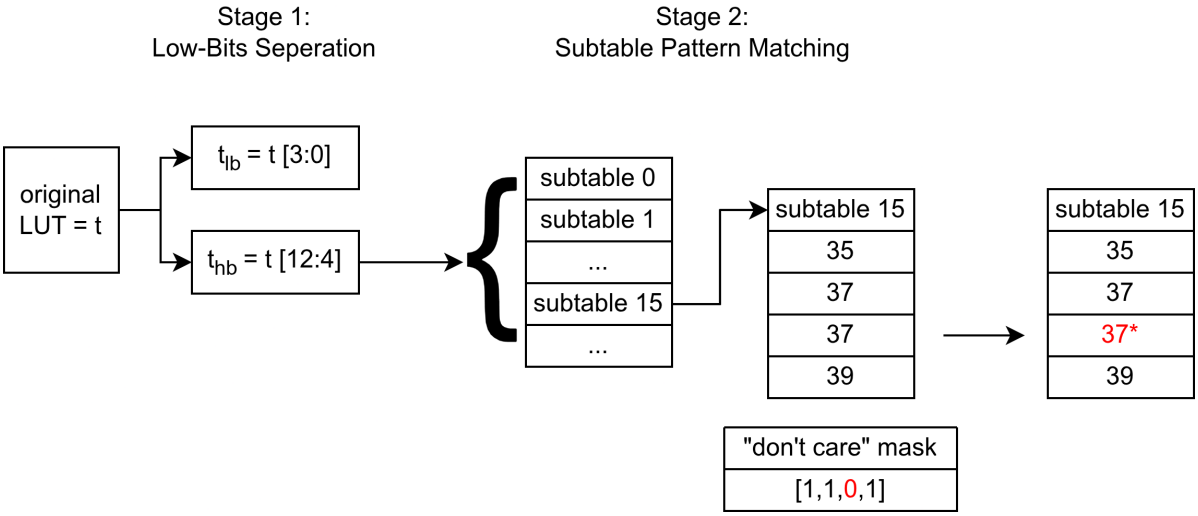


Figure A.1: Part 1 of the adaptation of the ReducedLUT algorithm.

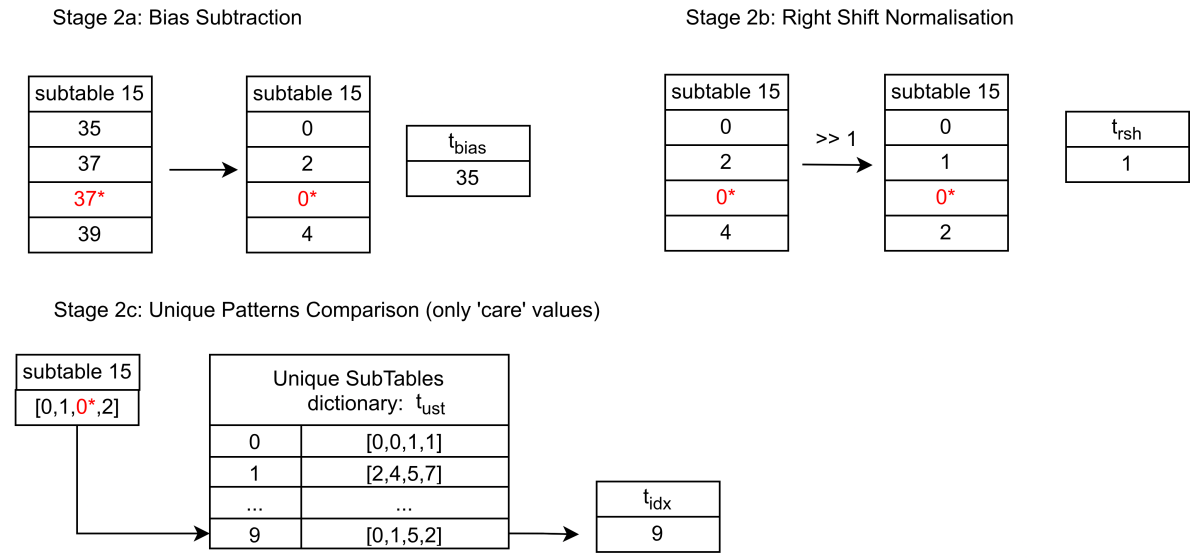
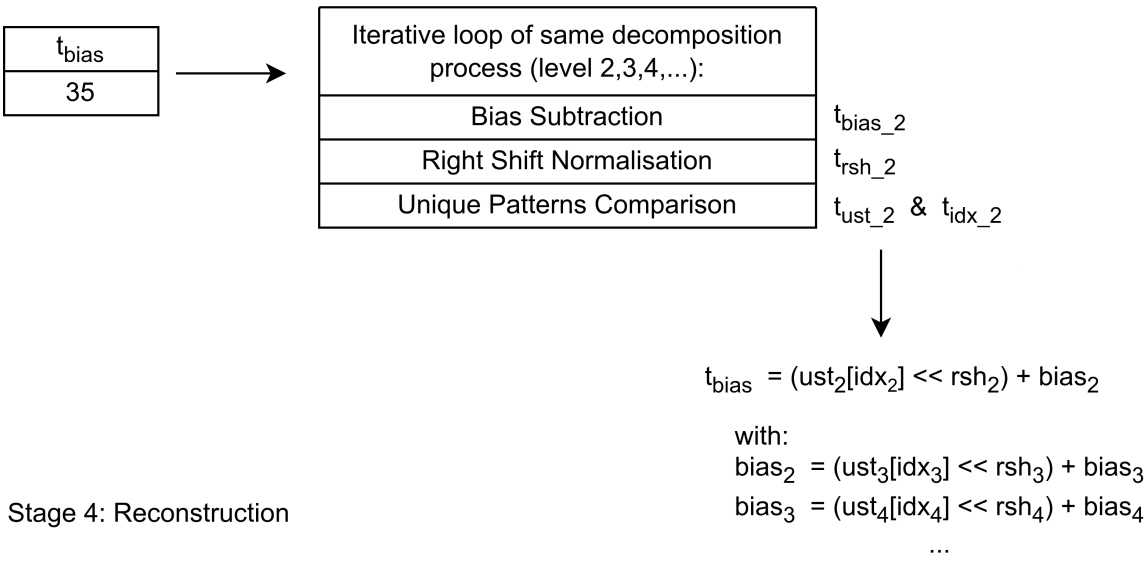


Figure A.2: Part 2 of the adaptation of the ReducedLUT algorithm.

Stage 3: Multilevel Bias Decomposition



Stage 4: Reconstruction

$t_{original} = \{(ust\ [idx] << rsh) + bias, lb\}$

Example:
$t_{15} = \{(ust\ [9] << 1) + 35, lb\}$ with "don't care" mask: [1,1,0,1]
$= \{([0,1,2^*,4] << 1) + 35, lb$

Figure A.3: Part 3 of the adaptation of the ReducedLUT algorithm.

B

Simulation results

The text in these sections needs to be adjusted; the focus now is just on placing the pictures here.

B.1. Dataset

B.1.1. Training Dataset

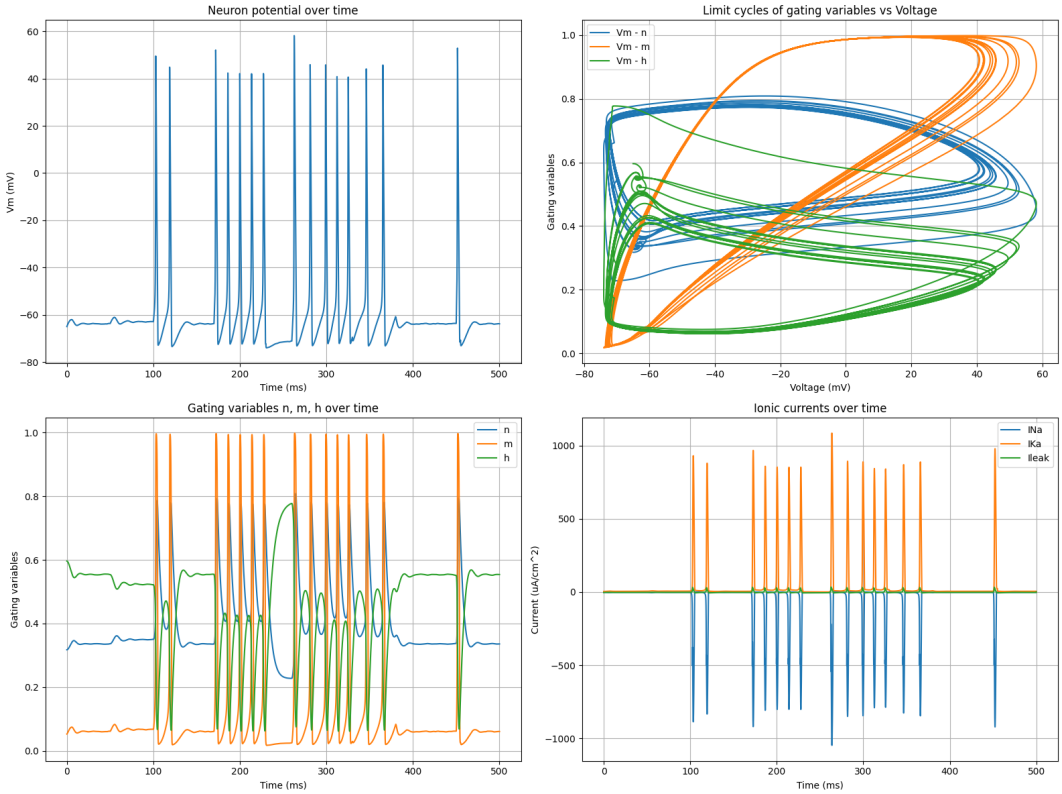


Figure B.1: Simulated Hodgkin-Huxley model showcasing the evolution of all intermediate and resulting parameters as a result of the stimulus input from Figure 5.2a

B.1.2. Validation Dataset

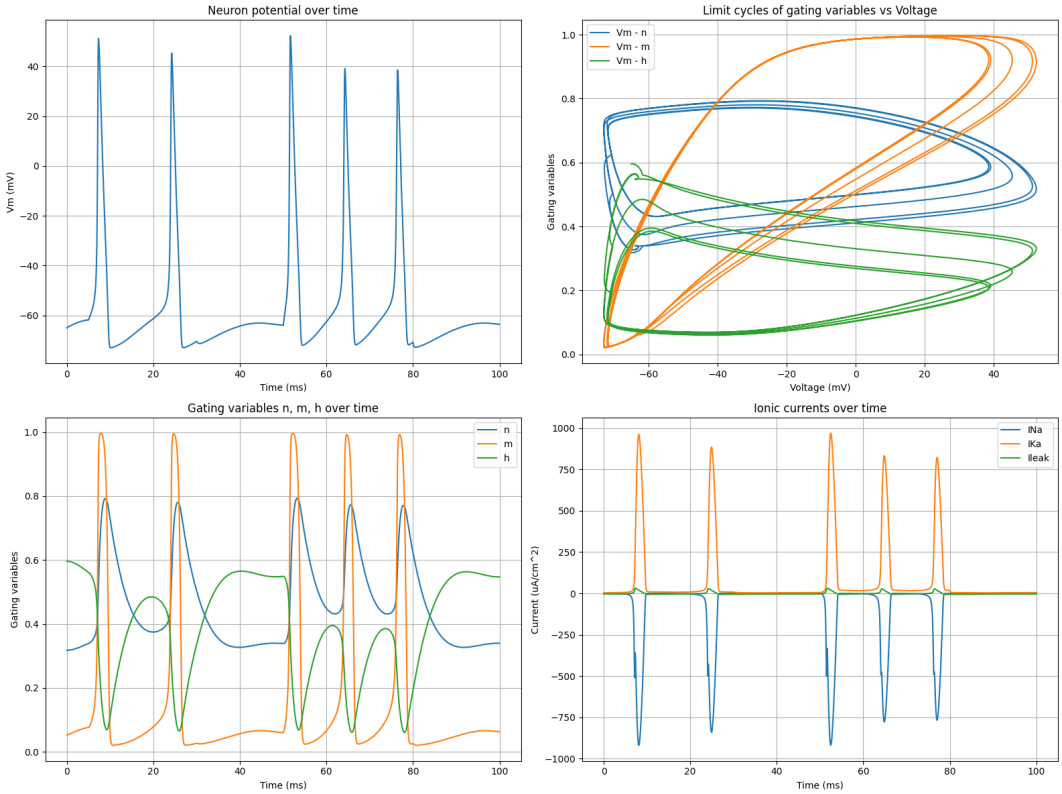


Figure B.2: Simulated Hodgkin-Huxley model showcasing the evolution of all intermediate and resulting parameters as a result of the stimulus input from Figure 5.2b

B.2. Gate Channel Normalisation Accuracy

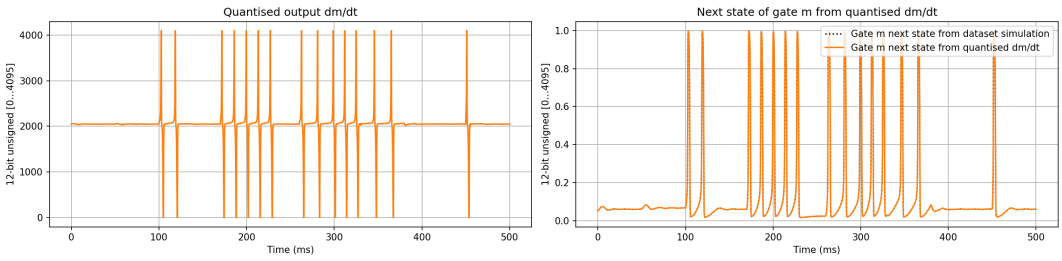


Figure B.3: Simulated LUT output of gate m channel LUT with a resulting plot from the quantised derivative output, dm/dt normalised by -1..+1

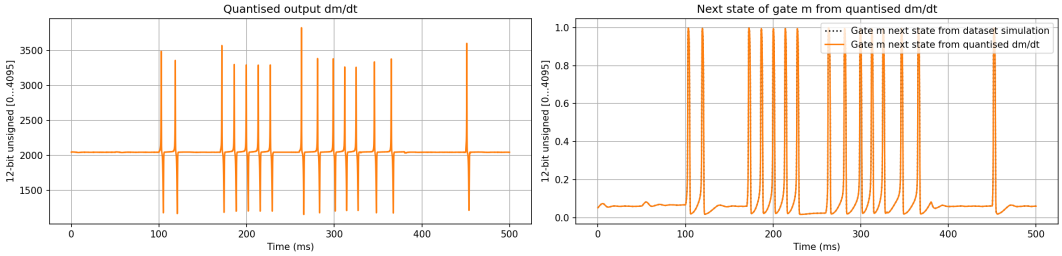


Figure B.4: Simulated LUT output of gate m channel LUT with a resulting plot from the quantised derivative output, dm/dt normalised by -3..+3

B.3. Reduced Bit Resolution

Comparison between three simulations that illustrate the effect of a reduced bit resolution on the LUT on which only Optimisations 1 and 2 have been applied; from top to bottom comparing between 12 bits, 10 bits and 6 bits.

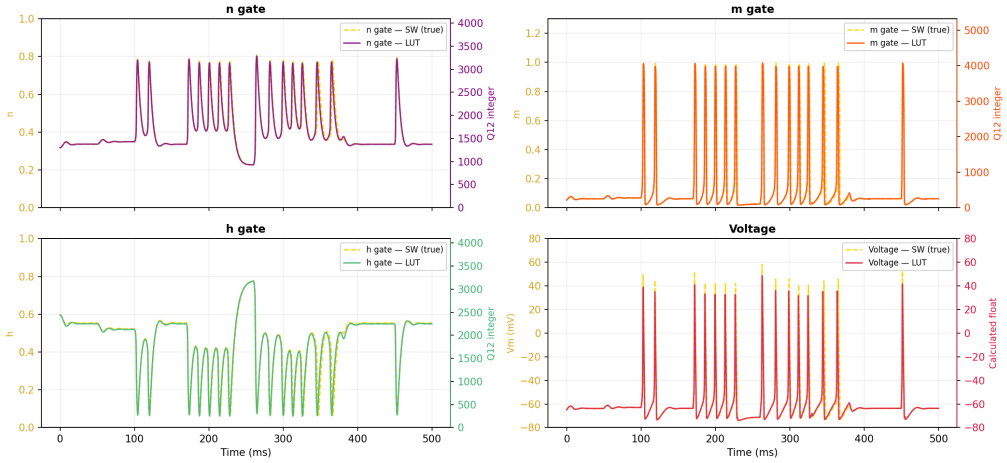


Figure B.5: 12-bit resolution

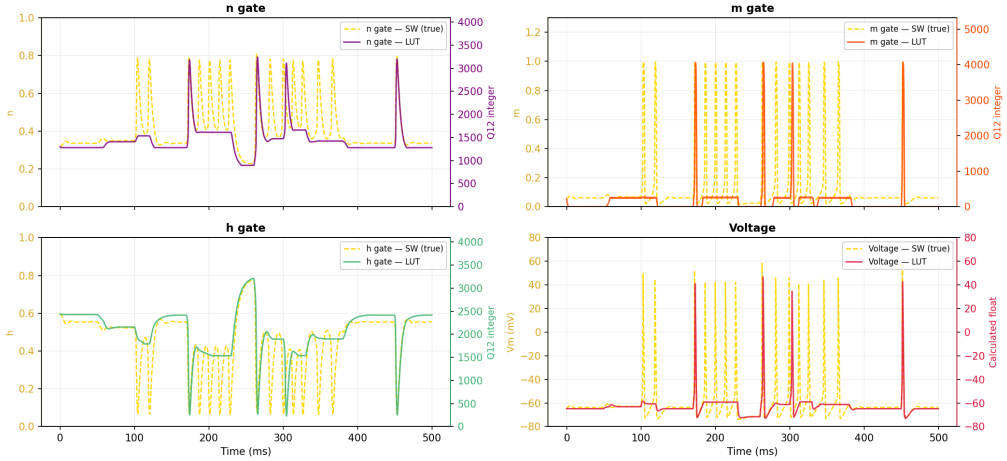


Figure B.6: 10-bit resolution

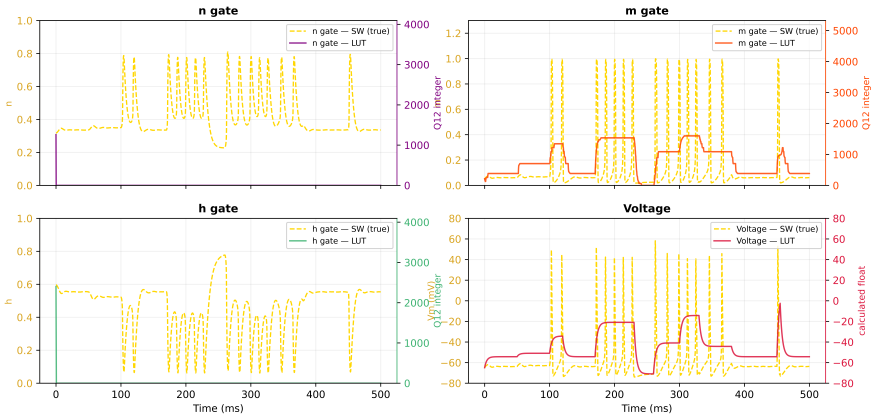


Figure B.7: 6-bit resolution

B.4. Neural Network Training Simulation

B.4.1. Optimisation 3: Gate NN

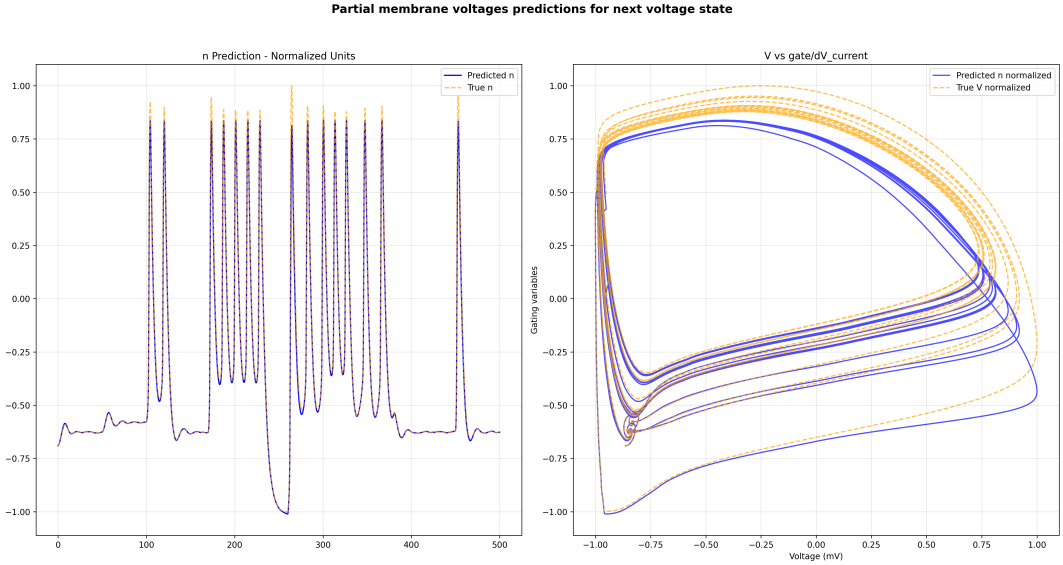


Figure B.8: Simulated NN autoregressive simulation gate channel n, 12 bits, normalised in own range

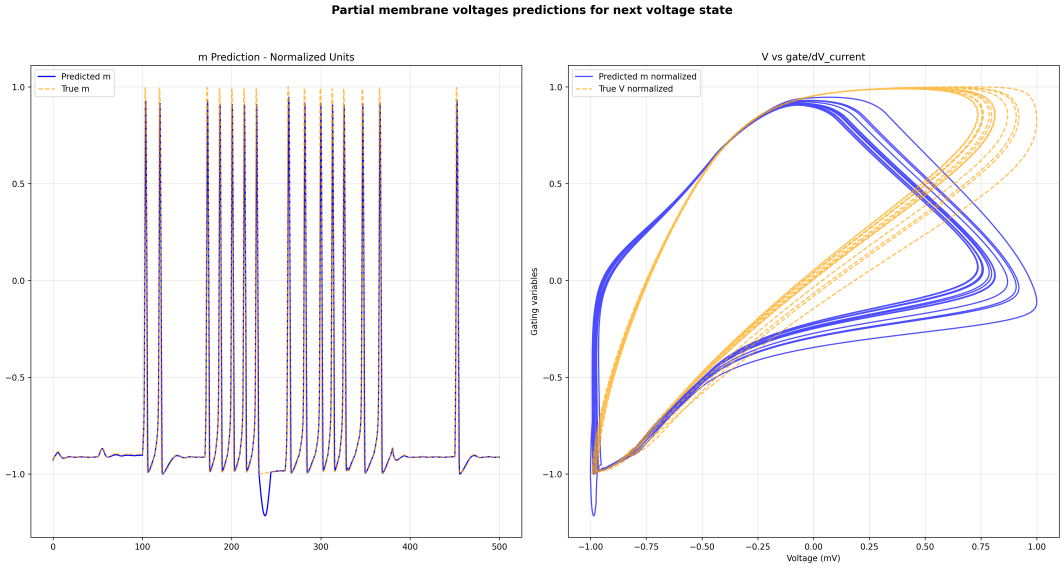


Figure B.9: Simulated NN autoregressive simulation gate channel m, 12 bits, normalised in own range

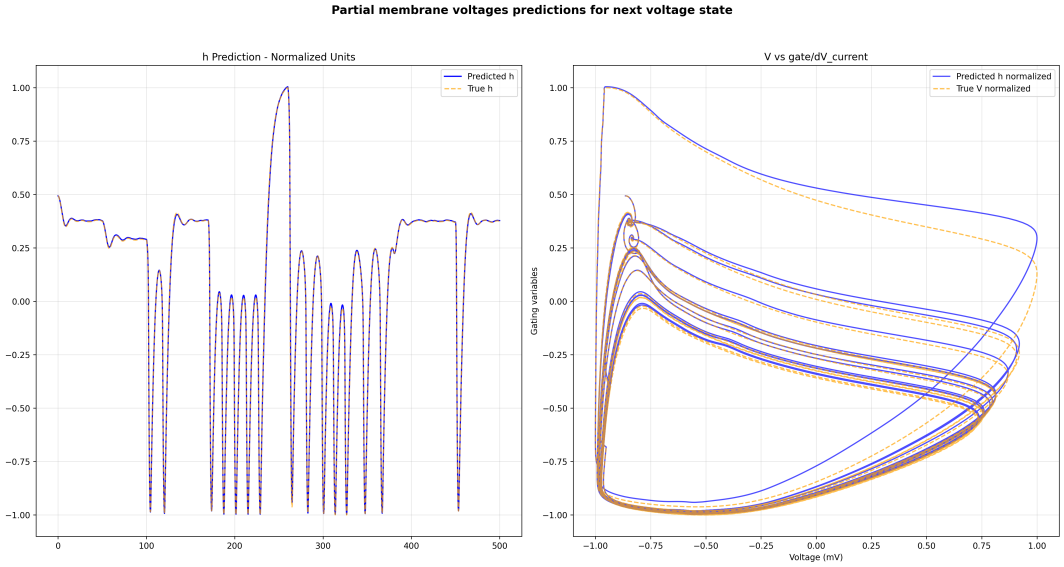


Figure B.10: Simulated NN autoregressive simulation gate channel h, 12 bits, normalised in own range

B.4.2. Optimisation 4: Gate NN

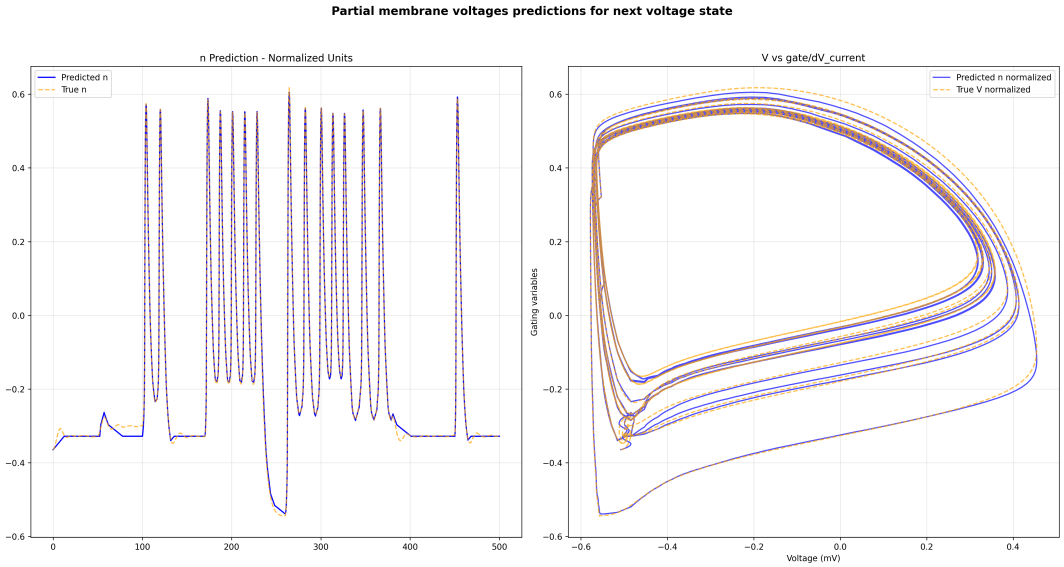


Figure B.11: Simulated NN autoregressive simulaiton gate channel n, 6-6-12 bits, normalised 0..1

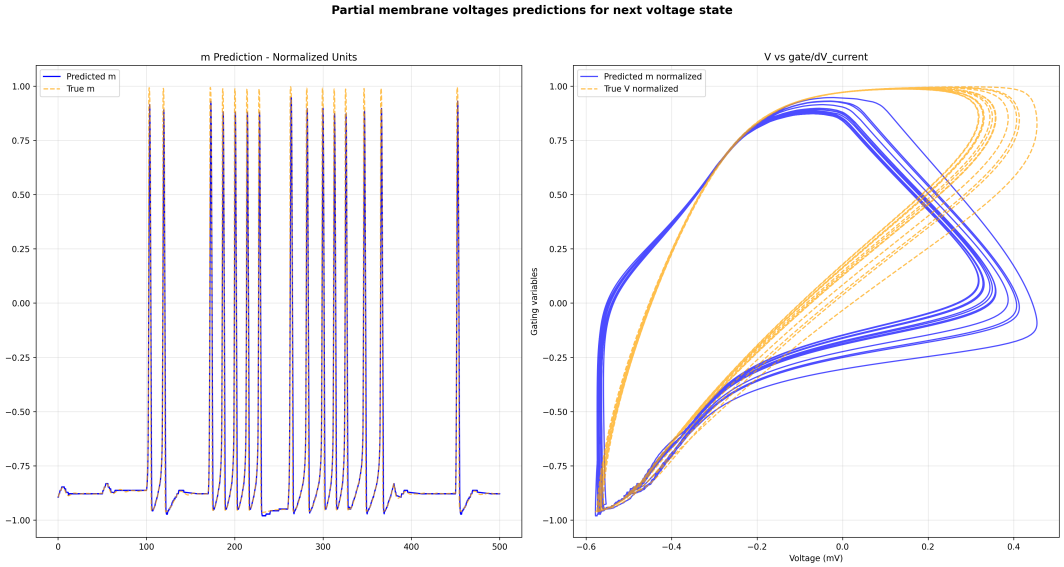


Figure B.12: Simulated NN autoregressive simulaiton gate channel m, 6-6-12 bits, normalised 0..1

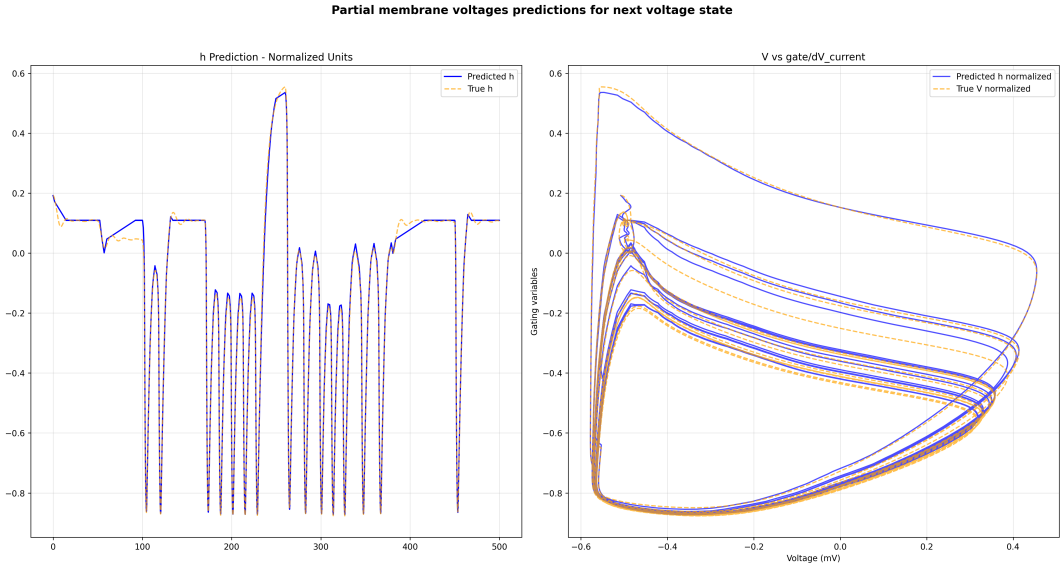


Figure B.13: Simulated NN autoregressive simulaiton gate channel h, 6-6-12 bits, normalised 0..1

B.4.3. Optimisation 4: IK and INa NN

design 3, so IK and INa:

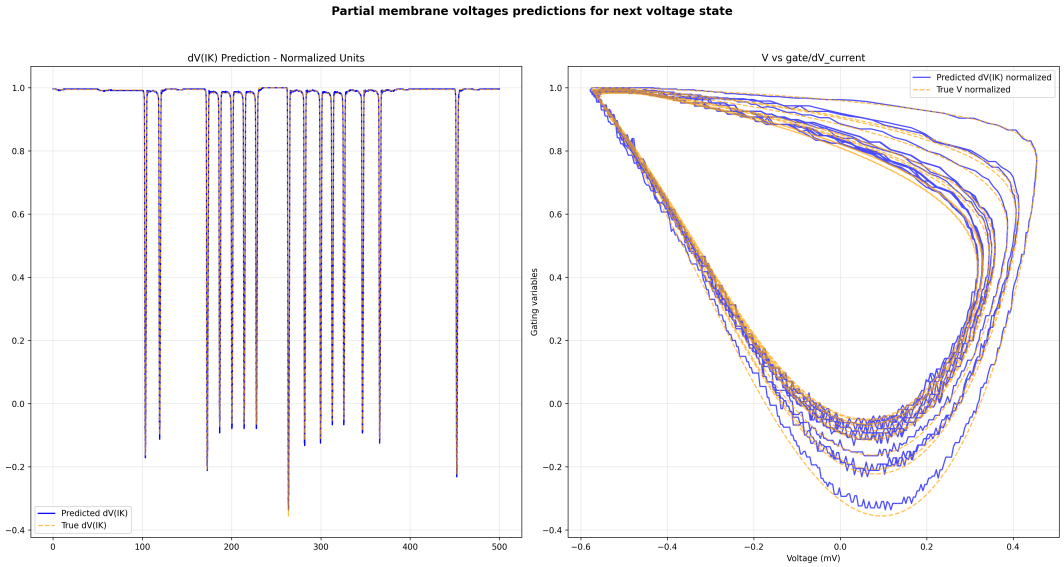


Figure B.14: Simulated NN autoregressive simulaiton IK, 8-6-12 bits, normalised -16..0

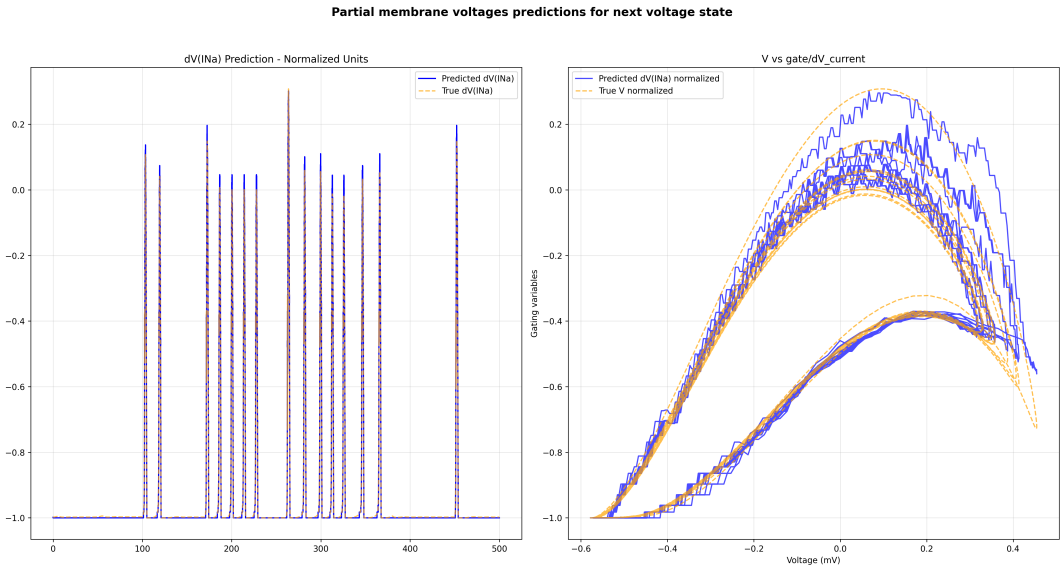


Figure B.15: Simulated NN autoregressive simulation INa, 8-6-12 bits, normalised 0..+16

design 3 + design 4: combined luts with n,m,h,Ileak,IK being NN luts. and INa being gm*m3 lut + dsp's:

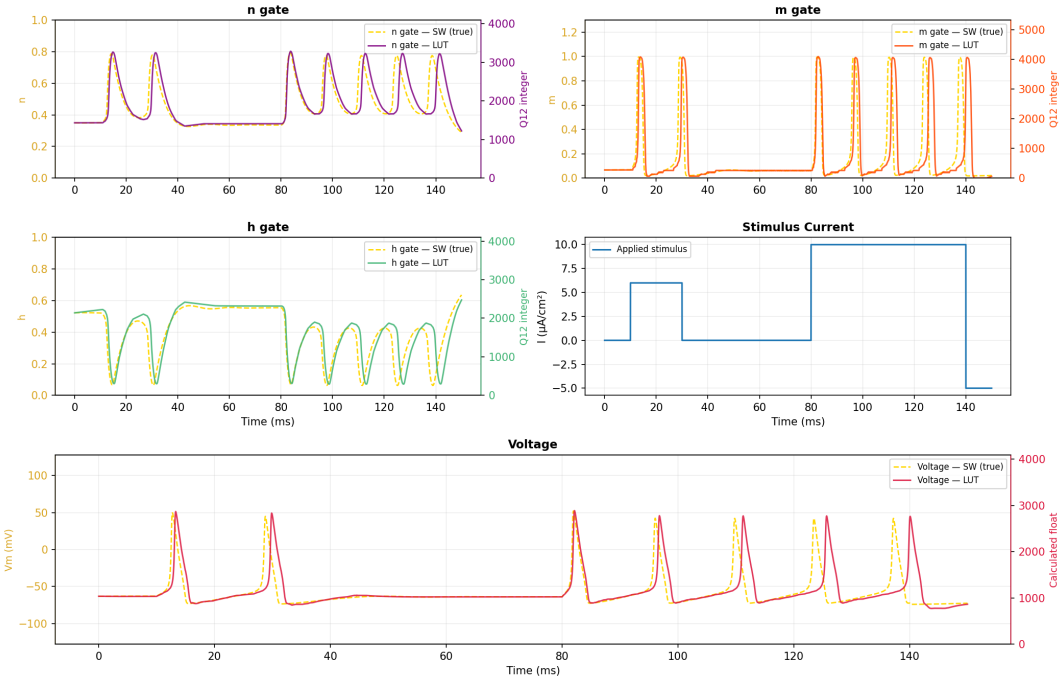


Figure B.16: Combined LUTs merging Design 3 and 4: gates n,m,h and IK & Ileak are Neural-Network-derived LUTs, while INa is implemented as the simple $g_{Na}m^3$-LUT and 2 DSP blocks

B.4.4. Optimisation 4: Voltage NN

design 2. input 12 bits, hidden 8 bits, output 12 bits

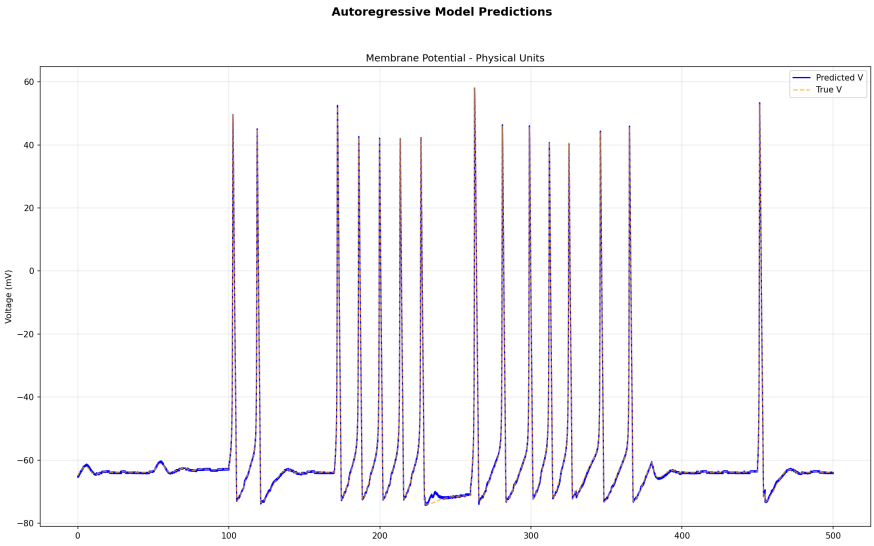


Figure B.17: Simulated NN autoregressive simulation of $n, m, h, V, I \rightarrow \frac{dV}{dt}$

B.5. Neuronal Network

B.5.1. Design 3

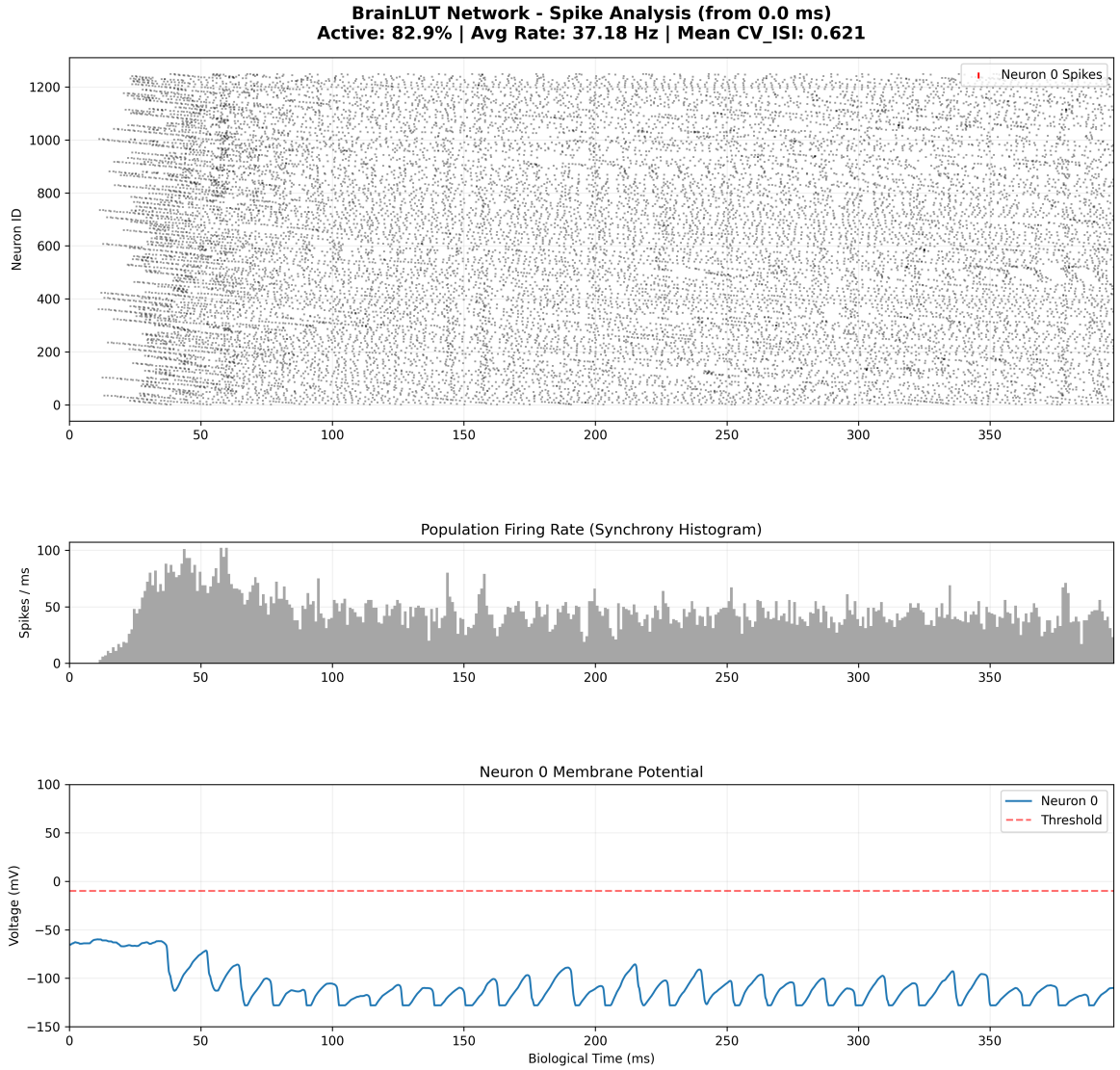


Figure B.18: Neuronal network simulation using the Design 3 [IONIC-SUM] core for 1250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.1 to -0.4. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference failed neuron (ID=0) (bottom).

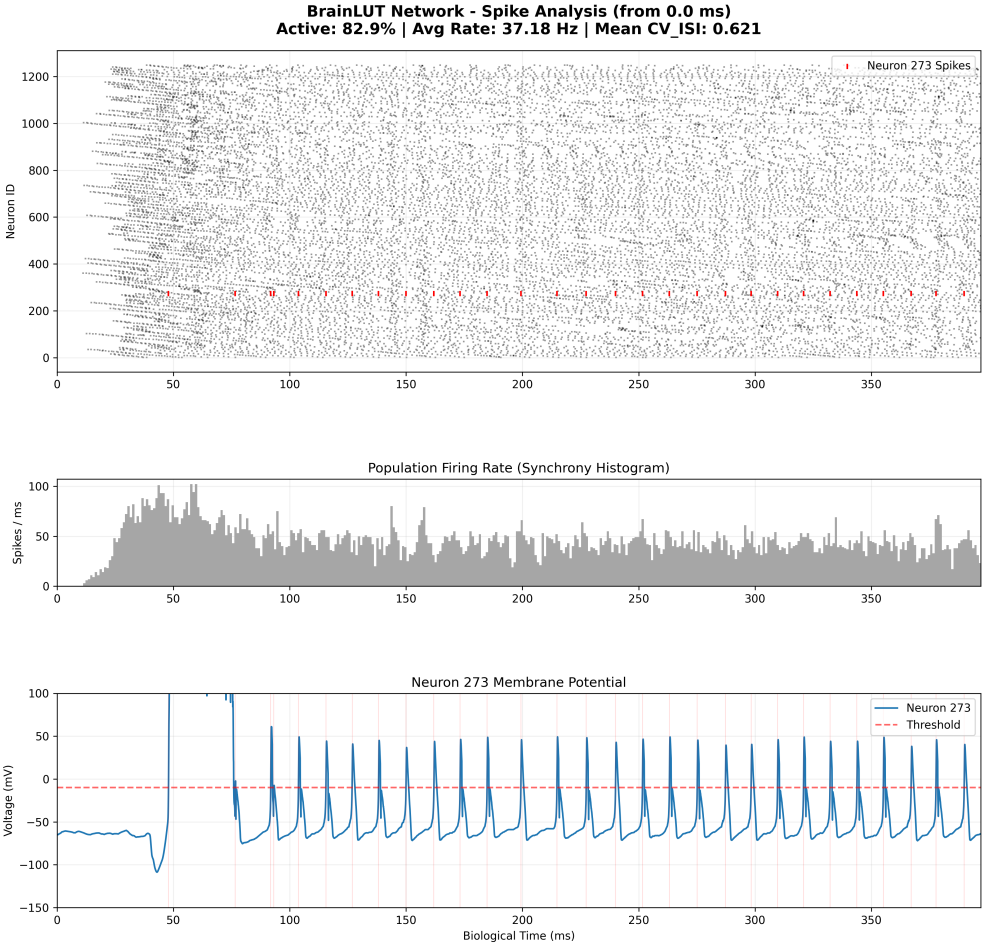


Figure B.19: Neuronal network simulation using the Design 3 [IONIC-SUM] core for 1250 neurons at real-time, 125 MHz processing speed, with an 80%-20% excitatory-inhibitory ratio and synaptic weights of 0.1 to -0.4. The figure displays a raster plot of spiking activity (top), a histogram of population firing rates (middle), and the membrane potential of a single reference neuron (ID=273) (bottom).