Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

CAS-2017-1532626

M.Sc. Thesis

Digital Neuron Cells for Highly Parallel Cognitive Systems

Haipeng Lin

Abstract

The biophysically-meaningful neuron models can be used to simulate human brain behavior. The understanding of neuron behaviours is expected to have prominent role in the fields such as artificial intelligence, treatments of damaged brain, *etc.* Mostly, the high level of realism of spiking neuron networks and their complexity require a considerable computational resources limiting the size of the realized networks. Consequently, the main challenge in building complex and biologically accurate spiking neuron network is largely set by the high computational and data transfer demands. In this thesis, I implement several efficient models of the spiking neuron with characteristics such as axon conduction delays and spike timing-dependent plasticity in a real-time data-flow learning network. With the performance analysis, the trade-offs between the biophysical accuracy and computation complexity are defined for the different models. The experimental results indicate that the proposed real-time data-flow learning network architecture allows the capacity of over 1,188 (max.6,300, depending on the model complexity) biophysically accurate neurons in a single FPGA device.

Digital Neuron Cells for Highly Parallel Cognitive Systems

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Haipeng Lin
born in Zhe Jiang, China

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Digital Neuron Cells for Highly Parallel Cognitive Systems "** by **Haipeng Lin** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: May 10th 2017

Chairman: _____
prof.dr.ir. Alle-Jan van der Veen

Advisor: _____
dr.ir. Ren van Leuken

Committee Members: _____
dr. Carlo Galuzzi

_____
dr.ir. Arjan Van Genderen

_____
dr.ir. Amir Zjajo

Abstract

The biophysically-meaningful neuron models can be used to simulate human brain behavior. The understanding of neuron behaviours is expected to have prominent role in the fields such as artificial intelligence, treatments of damaged brain, *etc.* Mostly, the high level of realism of spiking neuron networks and their complexity require a considerable computational resources limiting the size of the realized networks. Consequently, the main challenge in building complex and biologically accurate spiking neuron network is largely set by the high computational and data transfer demands. In this thesis, I implement several efficient models of the spiking neuron with characteristics such as axon conduction delays and spike timing-dependent plasticity in a real-time data-flow learning network. With the performance analysis, the trade-offs between the biophysical accuracy and computation complexity are defined for the different models. The experimental results indicate that the proposed real-time data-flow learning network architecture allows the capacity of over 1,188 (max.6,300, depending on the model complexity) biophysically accurate neurons in a single FPGA device.

Acknowledgments

I would like to thank my advisor dr.ir. Ren van Leuken for his assistance during the writing of this thesis, and Carlo, Amir, Arjan, who also gave me a lot of help. Without them, this thesis would not have been possible.

Haipeng Lin
Delft, The Netherlands
May 10th 2017

Contents

List of Figures

List of Tables

1 Introduction

The spiking neural networks (SNNs) [42] replicate the dynamic behaviors and information processing mechanisms of a biological neural system [39], and exhibit temporal pattern processing [33] and fault-tolerant capabilities [11]. Subsequently, the main challenge in designing complex and biologically accurate SNNs is primarily set by the high data transfer and computational demands. Nevertheless, it is only through largescale networks and/or real-time simulation that biological dynamics for specific experiments e.g. brain-machine interfaces, can suitably be modeled. Execution of such networks on CPUs with generic programming suites or neuromodeling-specific languages, however, require a prohibitive amount of time to complete. Field-programmable gate arrays (FPGAs), although slower than custom-made ASICs, due to the inherent high parallelism, are capable of providing enough performance for real-time and even hyperreal-time neuron network simulations. Additionally, via (partial) reconfigurations of the hardware, various neuron models, (e.g. Hodgkin-Huxley model [13], Integrate-and-Fire model [30], and Izhikevich model [28, 29]), as well as different network topologies and cell interconnect schemes can be simulated. This flexibility is substantially enhanced by the use of high-level synthesis tools (e.g. Vivado HLS), which speed up the development process.

In this thesis, I implement three models of the spiking neurons representing different trade-offs between the biophysical accuracy and computation complexity, whose characteristics such as axon conduction delays, spike timing-dependent plasticity (STDP) [21], electrochemical state descriptions [16], etc, are implemented in a real-time data-flow learning network. These models are Hodgkin-Huxley model, Integrate-and-Fire model, and Izhikevich model, respectively. Each one of them can be used for the different purpose in the network. Integrate-and-Fire model is one of the most simple neuron model in the world, which has only one variable. It is employed to investigate the maximal amount of neurons stably working in the learning network. On the contrary, Hodgkin-Huxley model, one of the most biophysically accurate neuron models [13], has high complexity of computation (e.g. huge exponential computations). Thus, the simulated number of neurons is quite limited, due to the expensive requirement of computing and storing resources. Indeed, it is usually applied for the examination of the various research questions related to the interplay between ionic currents, synaptic integration, dendrite cable ltering and other issues on single neuron dynamic. Additionally, the system is set to support for the inspection of the responses generated by the neurons in the network, which depends on the input signals and their own parameters. Izhikevich concludes 20 properties of neurons related to the spike train, and compared them among the different neuron models [27]. Although Hodgkin-Huxley model can simulate 18 properties in theory, it is not appreciate for the research on the spike train, due to the difficulties in specifying the suitable stimuli and ranges for the tens of parameters. Instead, Izhikevich model is a better choice, which can exhibit almost 20 properties. In

comparison with Hodgkin-Huxley model, it just needs to specify the correct values for 4 parameters, to output the expected spike train in the network.

In the system, the input is localized for each neuron cell, i.e. the input signals can be transmitted to the specific neuron cells at the same cycle. Concurrently, the parameters are also localized for each cell, when the neurons are initialized. Additionally, the implemented system offers configurable on- and off-chip communication latencies as well as neuron calculation latencies.

With the tool of Vivado HLS, (partial) reconfigurations of hardware, as well as different network topologies and cell interconnect schemes, are simulated to find the optimal configurations of the three neuron models. The experimental results indicate that the proposed network architecture can implement over 1,188 (max.6,300, depending on the model complexity) biophysically accurate neurons in a single FPGA device.

1.1 Problem Description

Because the different neuron models have their own features (e.g. computations of spike, required parameters, etc), the system should be capable of recognizing different external inputs for corresponding neuron model and translating the external analog signal into digital signal. Subsequently, the types and amount of parameters of each neuron model are different, so the computations referring to parameters on each cycle should be localized in the modules, where each module corresponds to one particular neuron model. In total, three kinds of computational modules are designed (functioning as a plug-in component), and each specific module is independent from the other modules in the system. Depending on the application requirement, the corresponding computational module can be incorporated (plugged-in) into the general structure of the network.

One design constrain is the communication costs. The extended Hodgkin-Huxley model in [15] has a communication costs which grow linearly with the amount of neurons, e.g., multiple FPGAs (up to 8) can be connected without limiting the performance. Although the implementation of three neuron models in the same network increases complexity of architecture, the communication costs are still expected to grow linearly with the number of cells.

Additionally, as the computational speed of system is much faster than the human being's brain, multiple neurons can perform the calculations sequentially on each cycle. The appreciate amount of time shared neurons should be evaluated, in case that the sum of execution time exceed the real brain time.

The proposed learning network is aimed to be implemented on the FPGA Xilinx (xc7vx550tffg1927-2) device. Consequently, the cost-effective resource utilization is prioritized for the neuron models in the system.

1.2 Goals

The main goals of this thesis are:

1. The interface is designed to recognize different external inputs offered to each neuron model, and to translate these signals from an analog into a digital signal. The data in the form of packets is then transmitted over the network.

2. The input is localized for each neuron cell, *i.e.*, multiple packets can be transmitted to the specific neuron cells at the same cycle. Concurrently, the parameters are also localized for each cell. The localization of parameters can better simulate the biological meaningful models in a real circumstance. [15].

3. Several models of the spiking neurons representing different trade-offs between the biophysical accuracy and computation complexity are implemented in a real-time, data-flow learning network, as well as the characteristics of neuron models such as axon conduction delays, spike timing-dependent plasticity (STDP), electrochemical state descriptions, etc.

4. Using Vivado HLS to synthesize and validate the new network and implement it on the FPGA. Via (partial) reconfiguration hardware, as well as different network topologies and cell interconnect schemes, the optimal configurations are defined for the neuron models.

1.3 Contribution

The contributions of the work presented in this thesis are the following:

- Several models of the spiking neuron are implemented in a real-time data-flow learning network [15]. The input is localized for each neuron cell, i.e., multiple packets can be transmitted to the specific neuron cells at the same cycle. Concurrently, the parameters are also localized for each cell. The system offers configurable on- and off-chip communication latencies as well as neuron calculation latencies. All parts of the system are generated automatically based on the neuron interconnection scheme in use.

- To implement the proposed network architecture on the aimed FPGA. Meanwhile, the optimal configuration for the neuron models are specified.

1.4 Thesis Outline

- Chapter 2 introduces related neuron cell models and integration of these models in the network. The previous work in [15, 4], which is the base of this thesis, is also be presented in this chapter.

- Chapter 3 presents an overview of the system design, excluding details of implementation.

- Chapter 4 presents the details of implementation of proposed system omitted in Chapter 3.

- Chapter 5 presents an evaluation of implementation results.

- Chapter 6 concludes the thesis based on the experimental results and give some suggestions for the future work.

Background and previous model description

2

In this chapter, a short introduction to `Spiking Neural Network`(SNN) and related work are given [42]. Next, some basic brain information and several SNN neuron models are highlighted, and subsequently, the tools to implement these neuron models in the neural network presented. Finally, the summary of the architecture in [6, 15, 4], which forms a basic architecture for this thesis, will be given.

2.1 Spiking Neural Network

Spiking neural network (SNN) is classified as the third generation of neural network model, increasing the level of realism in a neural simulation [23]. Besides only simulating the frequency of spikes, the concept of time and shape of the spikes are also taken consideration into the models in SNN. The former can be better used to investigate the brain, and the latter is that the neuron in SNN only generates the spike when the membrane potential exceeds specific threshold. When a neuron files, it generate a spike and transmit it to the other neurons, which increase or decrease their potential according to the signals.

The SNNs replicate the dynamic behaviors and information processing mechanisms of a biological neural system, and exhibit temporal pattern processing and fault-tolerant capabilities. Consequently, it requires more computational resources, in comparison with artificial neural networks (ANNs). Field-programmable gate arrays (FPGAs), although slower than custom-made ASICs (do not allow to alter the neuron model after manufacturing), due to the inherent high parallelism, are capable of providing enough performance for real-time and even hyperreal-time neuron network simulations.

2.2 State of the Art

IBM produced a neuromorphic CMOS integrated circuit in 2014, which is named TrueNorth. The TrueNorth, is a multi-core processor network on a chip design, with 4096 parallel and distributed neural cores. Each one can simulate 256 programmable silicon, and the total amount of simulated neurons is over one million [26]. In turn, each neuron has 256 synapses. Subsequently, it just consumes 70 miliwatts, which is about 1/10000 the power density required by the conventional microprocessors.

The NepteronCore, has been implement on the Virtex 4 FPGA, with up to 400 physiologically realistic neurons of the Hodgkin-Huxley model [25]. A PC-FPGA interface is designed to adjust the parameters and on-line display basic synchronization measures, e.g. potential, spike time, etc.

In [32], a Piecewise Linear Approximation of Quadratic Integrate and Fire (PLAQIF) neuron model, is simulated and verified on a Virtex 5 FPGA. The net-

work consists of 161 neurons and 1610 synapses with 4210 times real-time speed, in which the real time is defined as 1 ms per simulation step. The PLAQIF is a simple SNN model, and only generates the spike when the membrane potential exceeds certain threshold. In comparison with the Hodgkin-Huxley model, none parameters on biologically-meaningful information are included in PLAQIF.

Another design on the Izhikevich model, is proposed in [19]. The authors present a parallel SNN accelerator for producing large-scale cortical simulation on the FPGA. This system performs synaptic processing in parallel, with run time proportional to the firing rate of the network. 64,000 neurons can be simulated on a single FPGA, with 2.48 times real time speed. Compared with a recent GPU accelerator, the spike delivery rate in the network can achieves 1.45 times faster.

In this thesis, several SNN neuron models with characteristics such as axon conduction delays, spike timing-dependent plasticity (STDP), electrochemical state descriptions, etc, are implemented in a real-time data-flow learning network, using a Virtex 7 FPGA. The real time rate is specified in the range from 50 us to 5 ms, depending on the types of neuron models. In comparison with the other designs above, this system can seamlessly switch the neuron models, according to the different purposes.

2.3 The Brain

The brain is the most complex part of human body. It can be divided into three parts (Figure 2.1): brainstem, forebrain and cerebellum [8]. Brainstem maintains the characteristics of life, including heartbeat, digestion, body temperature, sleep, breath and so on. This part can be further subdivided into 4 major parts: i) Medulla is at the bottom of brain and connected with spinal cord, and it controls heartbeat, breath, digestion, ii) Pons is involved in motor control and sensory analysis; some parts of pons are connected to cerebellum, involved in movement and posture, iii) Midbrain, is related with hearing, vision, eyemovement and body movement, and iv) Reticular system, is a mesh structure constructed by massive complex neurons; the functions of this part is about states of consciousness.

The forebrain consists of the cerebrum, thalamus, hypothalamus and limbic system. i) Cerebrum is the largest part of the human brain, associated with higher brain function such as thought and action. The cerebral cortex is divided into four sections, called "lobes": the frontal lobe, parietal lobe, occipital lobe, and temporal lobe. ii) Thalamus can be seen as interchange station in the brain. The spikes from cerebellum and brainstem will first stop in thalamus, form where these spikes will be transmitted to cerebral cortex. iii) Hypothalamus plays an important role in control of endocrine system and keeping metabolism normal. iv) Limbic system, often referred to as the "emotional brain", is found buried within the cerebrum.

Cerebellum has two hemispheres and has a highly folded surface or cortex. This structure is associated with regulation and coordination of movement, posture, and balance.

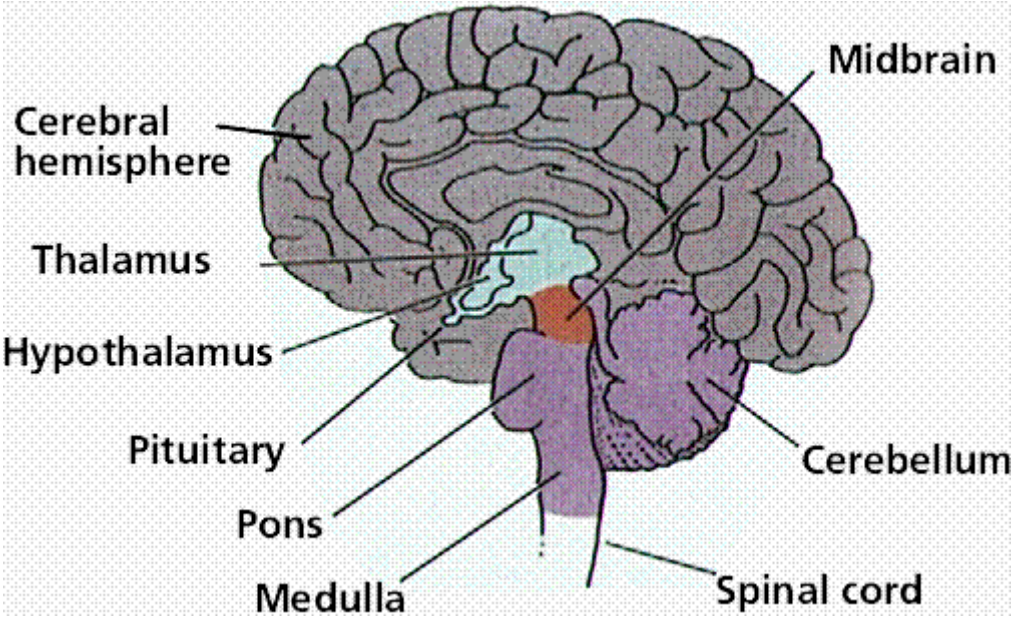


Figure 2.1: The brain structure [2]

2.4 Inferior Olivary Nucleus

The `Inferior Olivary Neurons`, (ION) situated in `Inferior Olive` (IO), are parts of the medulla oblongata and are located in the brainstem [6]. The IO is closely associated with the cerebellum, and is involved in control and coordination of body movement.

The IONs are connected with each other by `synapses`. Basic ION cell structure can be divided into three parts, `dendrites`, `soma`, `axon`. The `dendrites` receive the stimuli from the other IONs. These spikes will be transmitted to `soma` via synapse(s). Together with the states of neuron cell stored in `soma`, a response is given through `axon` (Figure 2.2). The `axons` leave medially through the hilum, cross the midline, and ascend into the cerebellum via the inferior cerebellar peduncle (climing fibers). Finally, they terminate by synapsing in the cerebellar cortex.

When the people want to move part of his/her body, a signal will be sent from `Celebrum`, to the `Cerebellum` through `Pons`. At the same time, ION will generate a action potential and send it via `axon` to `Cerebellum`, which is referred as climing fibers. The signal of desiring movement of body and action potential will be compared in the `Cerebellum`, to control and coordinate the desired movement of human body [43].

2.5 The Neuron Model

Izhikevich compared properties of neuron models and complexity of computations [28] (Figure 2.3). In this thesis, the three SNN neuron models are implemented in the neuron network.

2.5.1 The extended Hodgkin-Huxley Model

The extended `Hodgkin-Huxley model` is based on the `Hodgkin-Huxley model` [22], which is one of the most biophysically accurate neuron models [13]. The model consists of four equations and tens of parameters, including the membrane potential, activation

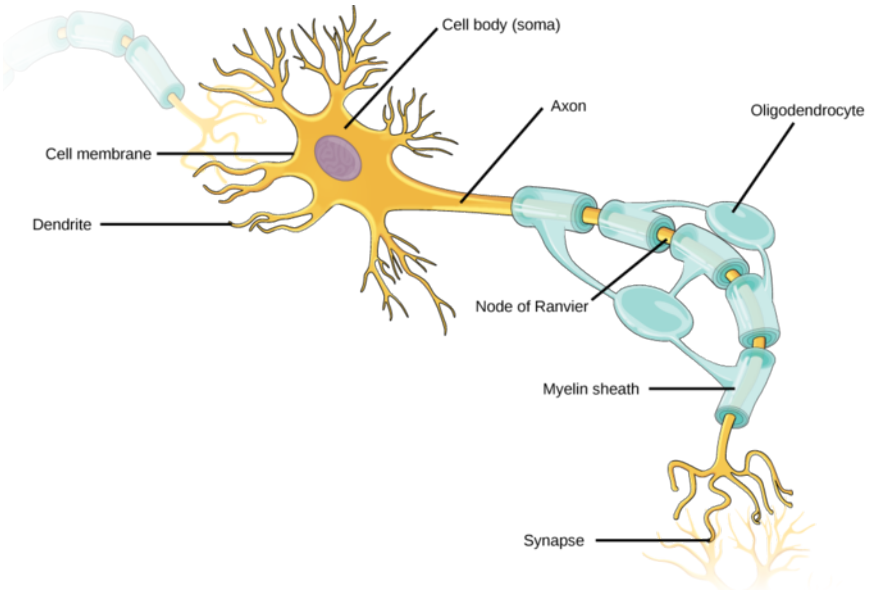


Figure 2.2: Structure of neuron [5]

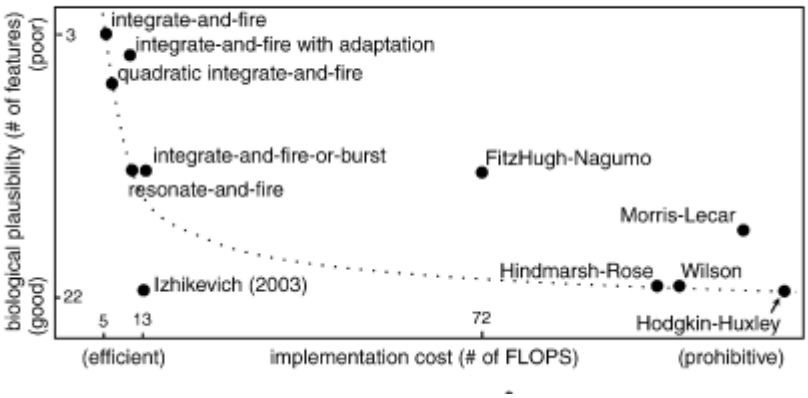


Figure 2.3: Implementation cost vs. biological plausibility [28]

of Na and K currents, and inactivation of Na current. The complexity involved allow examination of the various research questions related to the interplay between ionic currents, synaptic integration, dendrite cable filtering and other issues on single neuron dynamic.

The disadvantage of `Hodgkin-Huxley model` is the high complexity of computation. In other words, this model is extremely expensive to realized.*i.e.* huge exponential calculations are needed and approximately 1200 operations should be executed in 1 ms [28]. Thus, if the simulation time is considered, the number of neuron cells can be simulated is quietly limited.

2.5.2 Integrate and Fire Model

The `Integrate and Fire model` is one of the earliest and widely used neuron models, which is proposed in 1907 by Louis Lapicque [1]. The mechanism is quite simple: when the membrane potential reaches the threshold value, a new spike will be fired via

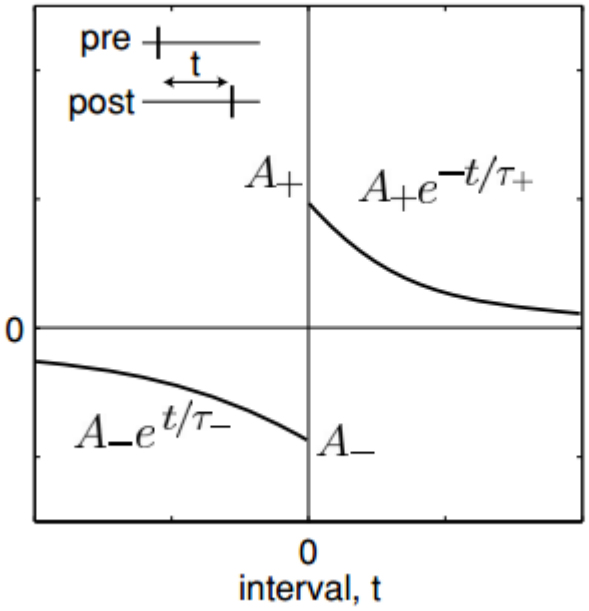


Figure 2.4: STDP Rule [29]

synapse to the neighbors of spiking neuron. Meanwhile, membrane of spiking neuron will be reset. The neuron of this model can fire tonic spikes with constant frequency.

Because `Integrate and Fire model` is relatively simple and has only one variable, only few properties of neuron can be exhibited. Thus, this model is usually employed to investigate the network of huge amount of neurons, if the biophysical meaningful features of the neuron cell is not an issue.

2.5.3 Izhikevich Model

The `Izhikevich model` is developed based on `Integrate and Fire model` [28, 29]. It can exhibit multiple types of properties of cortical neurons with the choice of specific set of parameters. Meanwhile, the computations are not very complex (about 12 operations needed in 1 ms simulation), in comparison with `Hodgkin-Huxley model`. As a result, it is efficient in large-scale simulation of cortical network.

2.5.3.1 Spike-timing-dependent Plasticity

The `Spike-timing-dependent Plasticity`(STDP) is a biological process that adjusts the strength of synapses among neurons in the brain (Figure 2.4). It adjusts the synaptic weights based on the relative timing of a specific neuron's input and output spikes [21]. The STDP partially explains the activity-dependent development of neuron models, especially with regards to long-term potentiation and long-term depression. The STDP rule is: if an input spike to a neuron occurs before that the neuron's output spike, then the input spike makes somewhat of synaptic weight stronger. On the other hand, if an input spike arrives on a neuron after the generation of spike, this particular input spike makes somewhat weak. The process continues until a subset of the initial set of synaptic weights remains, while the influence of all the others is reduced to 0 [38].

2.5.3.2 Delays

`Axonal Conduction delays` refers to time required for an action potential transmitted from `soma` to the terminals of `axon`, where synapses are formed with other neurons. The axonal conduction delays can vary greatly (depending on the axonal velocity and distance, respectively), and can be as large as 44 ms and as small as 0.1 ms [29].

2.6 Implementation Tools

2.6.1 SystemC

SystemC is a set of C++ classes and marcos that allows functional modeling of system, which can be compiled and executed as a normal C++ program. All the data types offered by C++, can also be used in SystemC. Additionally, some extra data types offered by SystemC library, as well as user defined, are available. A complex system design can be hierarchical decomposed into modules in SystemC. Among modules, this language provides an event-driven simulation interface, enable designer to simulate concurrent processes. The modules can be connected using ports and exports, passing simulated time and signals [14].

SystemC is a `Hardware Description Language`(HDL) with design capabilities at several levels: i) `Register Transfer Level`(RTL), is a design abstraction modeling a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. ii) `Behavioural Level`, describes the entire system with functions. iii) `Transaction Level`. In this thesis, the system first is modeled at behavioural level. Then the implementation of RTL will be generated automatically by `High level Synthesis` (HLS) tool [18].

2.6.2 Vivado HLS

`Vidado HLS` accelerates IP creation by enabling C, C++ and System C specifications to be directly targeted into Xilinx All Programmable devices without the need to manually create RTL.

The Vivado HLS synthesis a SystemC project can be implemented at the following steps. i) Importing a SystemC project, Vivado HLS will first automatically generate RTL simulation to validate design. ii) A VHDL description will be generated and co-simulated. iii) Using directives, Vivado HLS can explore different possibilities of hardware architectures. Comparing each performance, the best cost-efficient solution can be defined. iv) Finally, Vivado HLS will generate the FPGA netlist from previous VHDL description.

2.7 Previous work

The basis for the system of this thesis is in [15, 4, 6]. A system using SystemC was designed to simulate extended `Hodgkin-Huxley`. The external stimuli is converted from analog signal into digital signal in the module of input. Then this digital signals are encapsulated in packets and transmitted into network. The network in system is

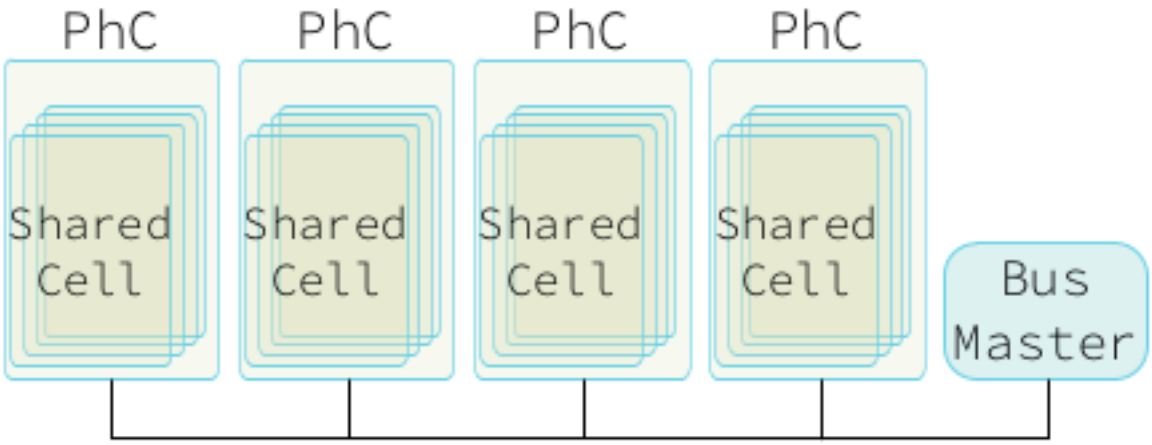


Figure 2.5: Overview of Structure of PhC [6]

constructed as the way of binary tree. The nodes in the tree consist of routers and clusters. Each router includes one upstream port and two downstream ports, where each port has its own routing table. When a router receive packet, it will look up its routing table to confirm whether this packet will be forwarded or not. Thus, the router can decide to forward the packet to next intermediate node or directly to destination of cluster, or to discard the receiving packet.

Communication cost among neuron cells is a key point affecting the overall performance of system. To reduce communication costs, time-sharing is introduced [6, 7] (Figure 2.5). Furthermore, multiple `Physical Cell`(PhC)s can be grouped into one cluster, sharing the same memory. These PhCs can access sharing memory in a Round-Robin order, and hence, communication latency between neuron cells in the same cluster can be significantly reduced. Consequently, the communication cost is proportional to the number of neuron cells whose computations and communications are localized in one cluster.

Based on the evaluation of the performance [15], this original simulator achieves the expectations. The communication cost increases approximately linearly with increase in the amount of the neuron cells. Additionally, this system can support the simulation over multiple FPGAs (up to 8), while growth of the communication cost is still kept linear. However, this design is only allowed to simulate the behaviors of the extended HH model at the simulation level using SystemC, while it can not be implemented in the FPGA with the tool of Vivado HLS.

To overcome the problem above, the work [4], based on the simulator [15], is proposed. It simplifies the design of the network in the system, and only focus on the implementation of single FPGA. The evaluation shows that this new design can implement maximal 1100 neurons of the extended Hodgin-Huxlay model in the FPGA.

Both systems are the costumed designs for the extend Hodgkin-Huxlay model. In today's world, tens of neuron models are developed, and each of them has its own biophysical features [28]. Designing the particular simulator for each neuron model, will be inconvenient and costs much time. So the requirement of a new system is needed, which can generally support for the simulations of multiple neuron models.

2.8 Summary

In this section, a brief description of SNN, related work, and basic biological brain information is given. The characteristics of three SNN neuron models, and the implementation tools, including SystemC and Vivado HLS are listed as well. At the end, a neuron network, which forms the basis for the work performed in this thesis is included.

3 System Design

This chapter presents the features of high-level system design. Firstly, requirements of designed system are listed, and then corresponding solutions are introduced. Finally, some adjustments are made to implement the designed system on a FPGA board, within time and resource constraints.

3.1 Requirements

In this thesis, the designed system should generally simulate multiple neuron models in the same architecture. Each neuron model is designed as an independent module using SystemC, and can be individually selected and plug in the system. Compared with a system with single neuron models [15, 4], this designed system encounters several important challenges.

- When given input stimuli, the neuron cell starts computing. For each neuron model, the values and frequencies of external stimuli, as well as the types of input data are different. Contrary to the `Integrate and Fire model` and the `Izhikevich model` where the current is given as a stimuli, `Hodgkin-Huxley model` receives the data in the form of potential difference.
- Each neuron model consists of a set of parameters. The biological meaning of each set of parameters are quite different. In `Hodgkin-Huxley model`, tens of parameters (19 parameters are defined in this system) record various states of neuron, such as membrane potential, activation of Na and K currents and so on. Sets of parameters in `Integrate and Fire model` and `Izhikevich model` decide the patterns of spike train. Once the value and frequency of external stimuli was defined, a specific set of parameters can result in a spike train according to one of 20 properties of neuron model [27].
- The characteristics of each neuron model vary significantly; the system should handle differences in the implementation of each model within the same network, while offering the possibilities for the independent modules of computations (*e.g.* Izhikevich model should implement axon conduction delay, while zero-conduction delay is set for the extended Hodgkin-Huxley model and Integrate and Fire model).
- When implementing the system on a FPGA board, resource and time constraints should be taken into consideration. With the tool of Vivado HLS, some directives can be added to optimize the design. Furthermore, localizing the individual design of neuron model as much as possible is a good solution to reduce resource cost, in case redundant parts take up too much hardware resources when switching within three models.

3.2 Input and Output

Any potential or current, which originates from the outside world, and represents the measurements of the physical quantity, is an analog signal. Before the stimuli is sent to the neuron cells, it should be converted to digital signal first, which is discrete time signal generated by digital modulation.

The interface connecting to external environment in this system can recognize both types of input (potential or current). It only focus on the value of signal and data format (e.g. fixed point, floating point), recognition of data types is determined manually while simulator switches to one specific neuron model. Once conversion of signal is done, the corresponding digital signal will be assigned with *id* and transmitted to destination of neuron cell in the network.

In the real world, a nerve system should receive multiple external stimuli concurrently. In order to be close to the real situation, this interface is capable of localizing input for each neuron cell. In each simulation cycle, the designed interface can deal with multiple external sources and send these packets with *id* one by one to the destinations. The *id* in packet is corresponding to a specific neuron cell in system, which is given at the initializing stage. After the packet is sent into the network, routers utilize the look-up routing tables to forward them to the next node. Finally, the cluster containing the destination of neuron cell will accept the packet and store it in the shared memory, so that the corresponding neuron cell can read it before calculation.

Additionally, the designed interface is also responsible to change the results of calculation back into analog signal, and write them to external files at the end of each simulated time cycle. Meanwhile, these results can be locally chosen to store. Once the range of selected neurons are defined in the configuration module, the other packets of result outside the range will be filtrated by the interface.

3.3 Parameters

Sets of parameters are locally stored in the memory of independent module for each neuron model. This design has two advantages. i) It is convenient to switch the neuron models in the network, otherwise the configuration module should be rewritten each time. ii) Because the calculations in the neuron need to read or write parameters at each cycle, localizing storage of parameters can significantly reduce latency, in particular communication cost.

For each neuron cell, the parameters are set individually (e.g. a set of parameters represents various states of neuron cell). Before the system start working, the parameters are initialized by several random functions. Thus, each neuron can have its own states. These functions specify the parameters within some certain ranges for the neurons, whose max and min values are pre-defined in the configuration file.

3.4 Scalability of Network

Nowadays, there are many different methods connecting components in the system. One popular method, which promises high performance and efficiency, as well as low

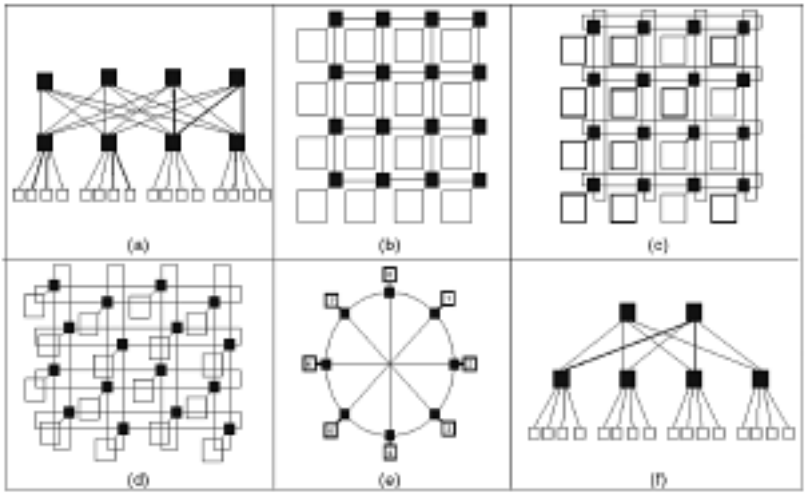


Figure 3.1: Typical Topology For Network [35]

resource cost, is Network-on-Chip (NoC) [35, 17]. Some typically topology of NoC are presented in Figure 3.1. In this thesis, a topology of binary tree is selected to implement the network (Figure 3.2), because the tree structure can be scaled up or down easily. In the designed network, leaf nodes represent clusters which contain multiple PhCs inside, while the other intermediate nodes are routers. Each router consists of *3* input and *3* output ports. One input (output) port is corresponding to be upstream in the tree, while the others are downstream. Routing tables are attached to each port, so that routers can forward the receiving packets, and avoid the sending packets to loop back. Additionally, several `FIFO` buffers are designed in the router to handle congestion. Each input or output port has a corresponding write- or read-buffer. Once write buffers are full, the following packets will be stored in the delayed buffer first. As soon as the write buffer is empty, the packets in the delayed buffer will be picked up and forwarded to destinations via write buffer. In order to simplify the design, all packets are defined with the same size and priority in the network. Hence, there is no need to split packets into several flits.

In this thesis, the amount of simulated neurons is dependent on three dimensions: the number of clusters, the amount of PhyCs in one cluster, and the time shared factor (the number of neurons allowed to work together in one PhyC). These parameters are all pre-defined in configuration file. Once system starts, it firstly initializes the network based on the parameters of configuration. A recursive function is designed to generate new branch from root node, until the number of leaf nodes is equal or larger than the pre-defined parameter. Consequently, by changing the parameters in the configuration file, the network can scale up or down conveniently.

3.5 Implementation of Neuron Models

The aim of this thesis is to design a system, which supports three SNN neuron models:

- `The extended Hodgkin-Huxley model`
- `Integrate and Fire model`
- `Izhikevich(2006) model`

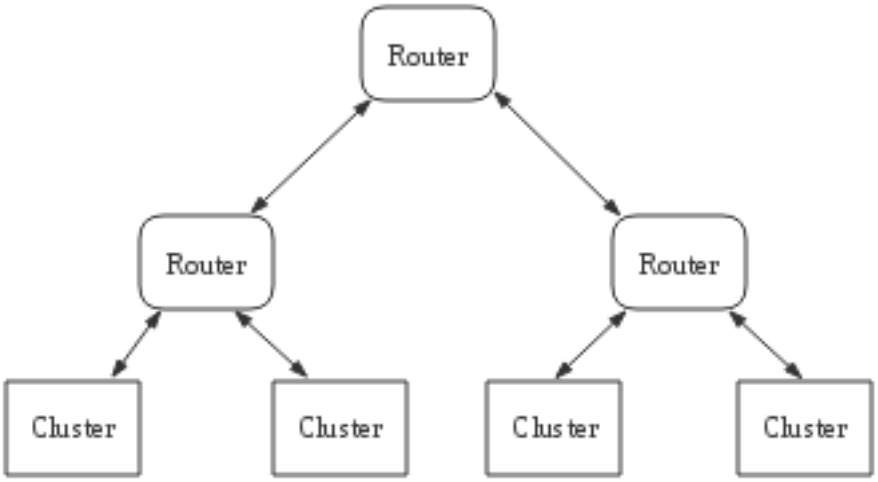


Figure 3.2: Structure of the Network

Implementation of the extended `Hodgkin-Huxley model` is based on [4, 15]. External inputs and parameters are localized to achieve the overall goals of system design. This means that multiple inputs can be accepted at each cycle and sent to the destinations individually. Additionally, each neuron cell is initialized with a set of parameters, where the values are different within some specific ranges. Calculation of `Hodgkin-Huxley model` can be separated into two parts [16]. The first half is computing potential of dendrite and internal current. At each simulated time step, component of dendrite will firstly access the share memory in a cluster, to read the potentials from its neighbors and from the external source (V_i). Once all these potentials are calculated, the potential difference can be used to generate the current as:

$$I_c = C * (0.8 * \sum_{i=0}^{N-1} (V_i * e^{\frac{-1*V_i^2}{100}}) + 0.2 * \sum_{i=0}^{N-1} *V_i) \tag{3.1}$$

where C is the conductance and N is the number of connected neighbours.

On the other hand, as long as receiving current via internal channel, the component of axon will generate a new potential of axon and update the parameters. At the end of cycle, new potential of dendrite will be stored in the local memory, and the axon potential will be sent upstream into network via the module of cluster.

Compared with `Hodgkin-Huxley model`, the computation of `Integrate and Fire model` is much more simple. The mathematical expression is below:

$$v_{new} = I + a - bv, \text{if } v \geq v_{thresh}, \text{then } v \leftarrow c \tag{3.2}$$

where v is the membrane potential, I is the input current, and a, b, c and v_{thresh} are parameters. When the membrane potential v reaches the threshold value v_{thresh}, the neuron will fire a spike, and v is reset to c.

In this thesis, a single neuron of `Integrate and Fire model` is designed to connect with $(n+m)$ nodes, where n is the amount of neurons sending the responses to the specific cell via the pre-synapse, and m are the ones receiving responses from the neuron via post-synapse. Both n and m are random values defined by a connecting matrix. This matrix is automatically generated using `Python` [37] and loaded by the system as a script (JSON) [20], when the simulator starts to work. Hence, in each cycle, a neuron

of `Integrate and Fire model` will detect whether there are some spikes coming from its neighbors or external sources first. Then, a new input current can be calculated, depending on the receiving spikes and external signals. Afterwards, this current will be taken into the function (3.2), resulting in a new potential of dendrite. If this new potential exceeded specific threshold, `soma` will generate a spike and pass it to all relative neighbours via post-synapses. Concurrently, the potential of dendrite is reset to a constant, which is specified by the parameter in the configuration file. On the contrary, if the new potential of dendrite can not satisfy the requirement generating a spike, it will be stored locally in the memory, waiting for the next round of calculation.

In the `Izhikevich model`, the idea of generating spikes are quite similar to the Integrate-and-Fire model. The differences are the parameters and equations, which are expressed below:

$$v = 0.04v^2 + 5v + 140 - u + I, \text{if } v \geq 30, \text{then } v \leftarrow c, u \leftarrow u + d \tag{3.3}$$

$$u = a(bv - u) \tag{3.4}$$

where variable v represents the membrane potential of neuron and u represents a membrane recovery variable, which accounts for the activation of K^+ ionic currents and inactivation of Na^+ ionic currents, and provides negative feedback to u.

Beside the characteristic of spikes, the neuron of `Izhikevich model` should also implement axon conduction delay and STDP [29]. In this thesis, the delay between two neighbors is randomly assigned from *0* to a pre-defined max value, and is stored in the local memory (local delay information). Because the generation of spikes and transmission to the neighbors are performed within the same cycle, the spikes will be stored in the delay buffers first. Subsequently, they are re-marked with actual arriving time, according to the local delay information. In each cycle, the neuron checks whether there are some spikes in delay buffer meeting the real injecting time. Hence, all suitable spikes will be fetched and used in spiking functions above.

Finally, the system adds a global memory in a top module to record and control synaptic weights for all neuron cells. This design is aimed to realize dynamic change of synaptic weights among neurons real-time conveniently. When a neuron generates a spike, it will store its own STDP parameters in the global memory and notify the neighbors. When the neuron is informed that some neighbors generating spikes, it will update the STDP parameters in the memory, according to the STDP functions. In Izhikevich model, the synaptic weights directly influence the amount of the current flow through the synapses, which is used to calculate the potential of dendrite. On the contrary, if no spike is fired between two cells, the relative synaptic weight will decay proportionally.

3.6 Synthesis

The last section describes synthesis of this SystemC project, so that it can be implemented on the FPGA. To increase the efficiency, several adjustments are made for the design: i) Arrays are placed in BRAMs to reduce the cost of hardware (LTUs and FFs).

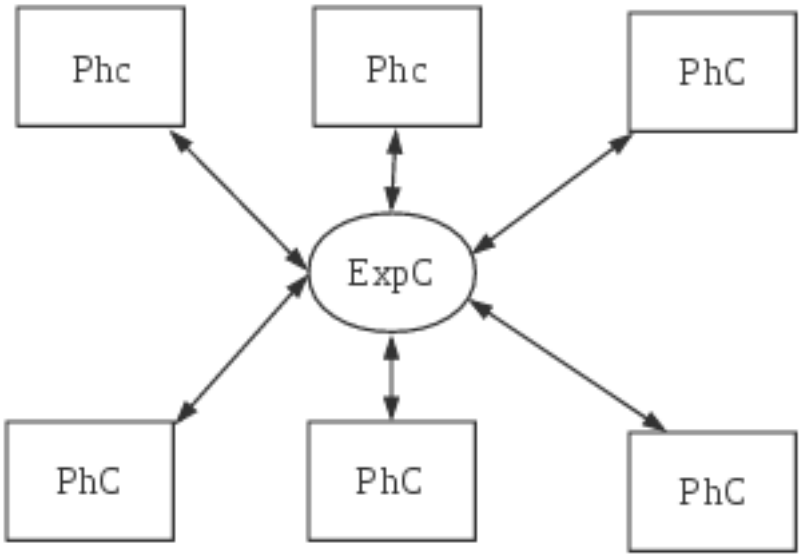


Figure 3.3: Several PhCs share one ExpC [4]

Arrays which should be accessed in parallel are separately placed in their own memory, while some other data, such as parameters are assigned the same BRAM, sharing a signal memory port. ii) Since several PhyCs shared the same memory in a cluster, some wait states are needed in the SystemC code so that no stalls occur during the computations of neuron cells. iii) The division calculation is rarely used since the unit of divider takes up quite a lot of hardware resource. By instructing scheduler to limit the amount of dividers, multiple neuron cells can share one single divider to perform the dividing functions, reducing hardware cost iv) In the general network, the hardware resource consumed by the design of individual neuron model will become redundant, when the system is switched to another model. Consequently, implementing characteristics of each neuron model, as much as possible, in the independent module, can significantly decrease the overall resource utilization in the system. v) Either single or double floating-point precision can be chosen in the system.

The extended Hodgkin-Huxley model involves multiple exponential computations. To save the resource costs, a particular sub-module named ExpC is designed to perform the exponential calculations [4]. In this thesis, several PhyCs of the same cluster can share one ExpC. The number of ExpCs in one cluster is determined by the time-share-factor, which is pre-defined in the configuration files. In each cycle, the neurons send the data, including exponent operands and their own addresses, to the corresponding ExpC in pipeline. Once the ExpC received the data, the exponent operands and address are stored separately in two FIFO buffers. After an exponent is calculated, the ExpC will read the address at the top of address buffer first. Subsequently, the data with specific address will be written back to the corresponding neuron via output buffer.

As there is only one exponential calculation in the Integrate and Fire model and none in the Izhikevich model, the sub-module of ExpC is removed in their independent modules to reduce the hardware costs. Instead, some additional resource utilization is required to perform the characteristics of delays, and STDP in the Izhikevich model. e.g. some more memories are required to store the delayed spikes, delay information,

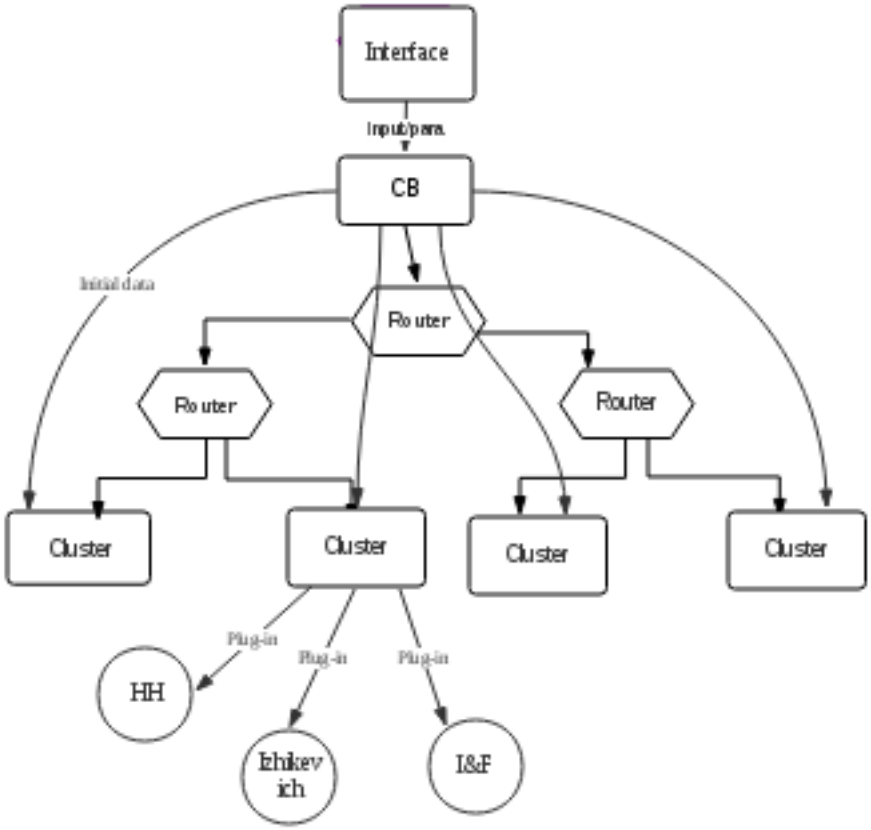


Figure 3.4: Overview of data flow as designed in chapter 3 and 4

and synaptic weights, *etc.*

This optimized system can find a suitable trade-off between the performance and resource cost. Using Vivado HLS, the FPGA netlist is finally generated automatically. (The data flow is in the Figure 3.4, and the overview of system is in the Figure 3.5).

3.7 Summary

This chapter proposed the requirements of system design in section 3.1, and then the corresponding designs are present. Section 3.6 introduces some adjustments, which helps the neuron network to find a cost-efficient solution of implementation on FPGA board.

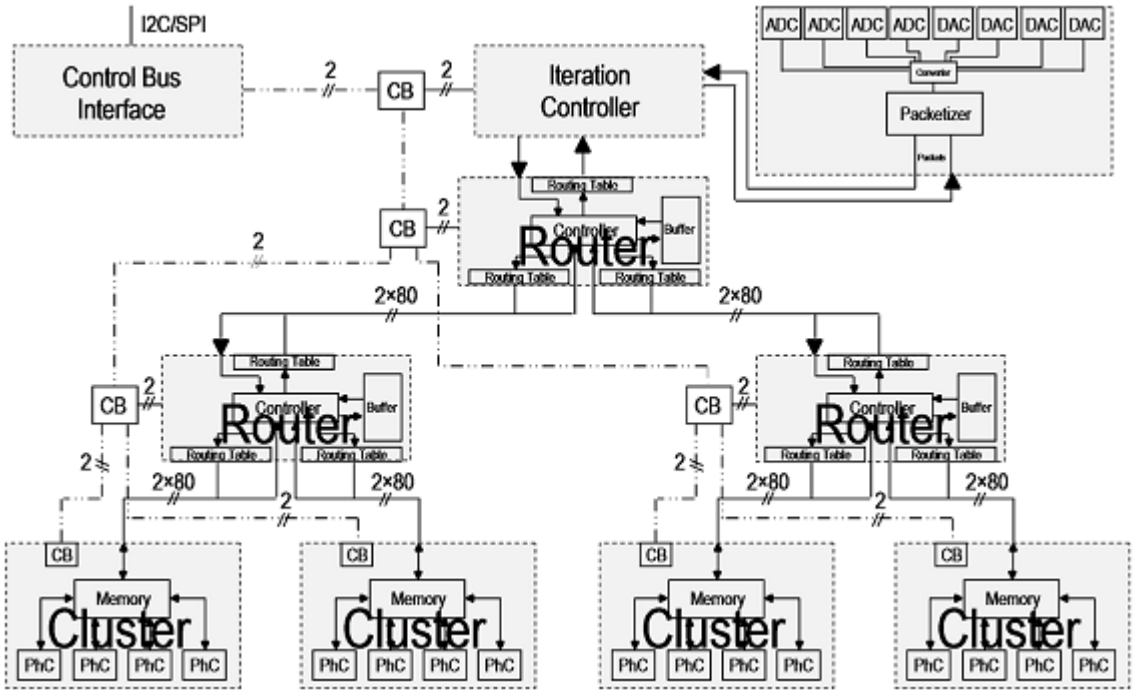


Figure 3.5: The NeuronNet system overview. The computing elements (the PhyCs) are grouped inside a cluster to make communication between neighboring cells fast. These clusters are connected in a tree topology NoC. The router fan-out in this case is 2, and can be changed according to the requirements of the implementation. The same holds true for the number of PhyCs in any cluster [15].

4 System Implementation

This chapter presents the details of system design derived in chapter 3. Several sections are given below, including interface connecting external sources, implementations of three neuron models, and optimized design for the synthesis.

4.1 Interface connecting to outside world

In this system, the interface is designed to read the signals from external sources, and write the simulation results back into the specific files. This input/output module consists of three sub-modules, ADC, DAC, and Inout (Figure 4.1).

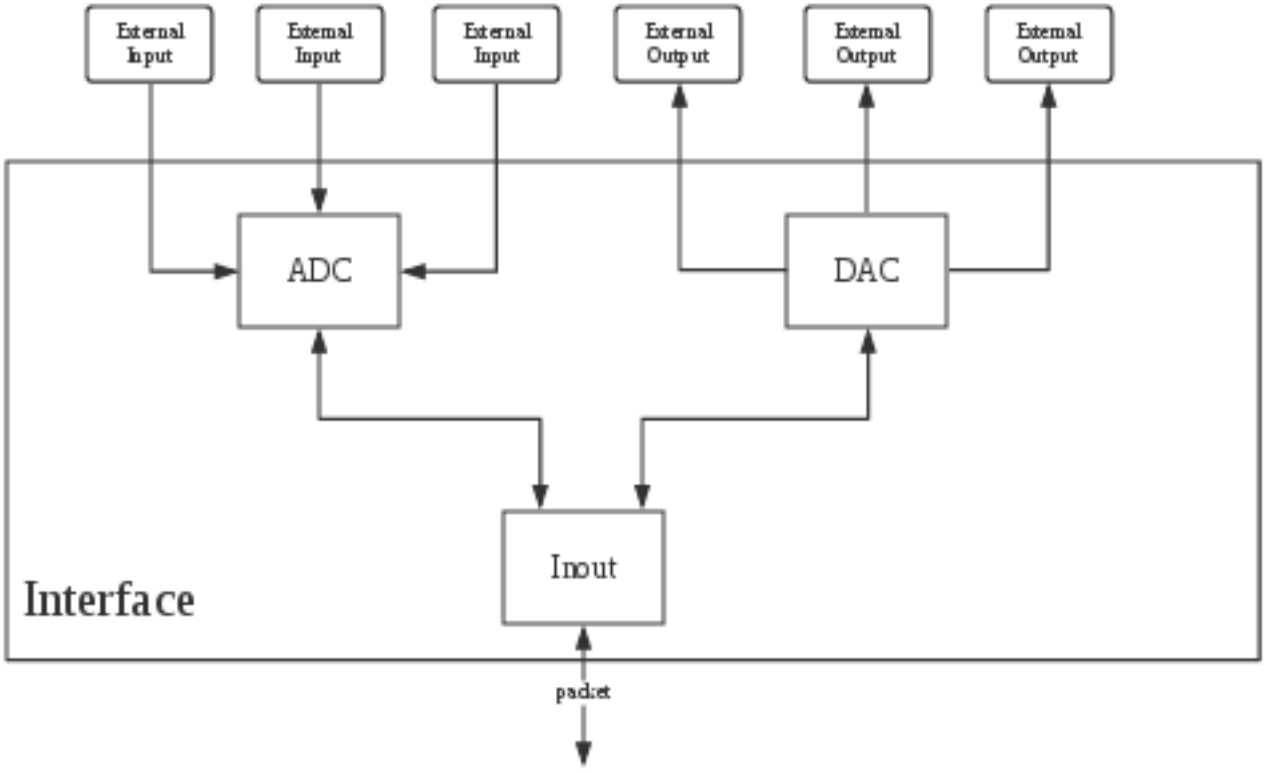


Figure 4.1: Data flow of external interface

```
 52.32,-30,-29,-25
 17,-12,21,22,-16
-52,-30,-29,-27
 12,-17,-21,22,8
-60,17,-31,2,12
-50,-30,-29,-14
```

Figure 4.2: Example of external inputs or outputs stored in the text file

4.1.1 Inputs and Outputs

The ADC is responsible for conversion of the arriving analog signals into digital signal. The external inputs are modeled, and pre-defined in a text file (Figure 4.2). Each line represents the injection of all outside data at one time step, and multiple external stimuli are separated by comma. On the contrary, the DAC writes the results back to the specific external sources.

When the module Inout receives data from the ADCs, it assigns the data with a unique *id* starting from $n + 2$ (n is the number of neuron cells in the system, and the *id* with n or $n + 1$ is set for the control of synchronization). Afterwards, the data is sent into network in the form of packet, with the same size and priority. On the other side, the packet containing result from network is extracted in the Inout. According to packet *id*, this result will be written back to the specific external address.

4.1.2 Locality of data

4.1.2.1 Localization of inputs

The localization of inputs means that multiple external signals can be sent to the specific neurons in each cycle. This adjustment improves the simulation of neuron model more close to the real situation in human brain (neuron network usually is given multiple motivation simultaneously).

A matrix of connectivity is defined before system starts, based on the parameters in configuration file. This matrix is a two-dimension ($N \times N$) array ($N = n_c + n_{input} + n_{output} + 2$, where 2 is a constant for the control of synchronization, n_c is the number of neuron cells, n_{input} and n_{output} represents the amount of external inputs or outputs, respectively). An example of matrix is given below:

$$a_{ij} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{4.1}$$

The $a_{ij} = 1$ means that the neuron i links to the neuron j, otherwise, 0 strands for no connection between the corresponding two neuron cells.

When initializing the network, the construction of routing tables are based on the matrix of connectivity. The range of *id* from $n_c + 2$ to $n_c + 2 + n_{inputs} - 1$ is assigned to the external inputs, each one is relative to a specific input address. By setting the value

to 1(0) in the matrix, a pair of input and neuron, can be connected (disconnected). Hence, the system is capable of specifying each input for the required neurons.

As mentioned in section 4.1.1, the *id* n_c or $n_c + 1$ is reserved for the control of synchronization. At the beginning of each cycle, the module of controller (control bus) will sent out the packets containing the *id* n_c to all clusters. Once the clusters received these packets, they will issue the new starting signals to their own neuron cells immediately. On the other side, the packets with *id* $n_c + 1$ are used to notify the controller that all clusters have already finished calculations, which are generated by each cluster. A count register is placed in the controller, in order to account for the number of arriving packets with *id* $n_c + 1$. As long as this count equals to the number of clusters, the controller will give out the new iteration command in the form of packet ($id = n_c$).

4.1.2.2 Localization of Outputs

The localization of outputs can helps the researchers to selectively investigate some specific cells. The *id* starting with $n_c + 2 + n_{inputs}$ belongs to the external outputs. When setting the value to 1(0) for a pair of neuron and output address in the matrix, the simulation result from the neuron can (not) be written back to the corresponding output files via network.

4.2 Implementation of The Neuron Models

The characteristics of the neuron models are already introduced in Chapter 2, and in Chapter 3 the basic ideas of system design are derived. In this section, the details of implementations in the network are present, which are divided into three parts, according to the three different neuron models.

4.2.1 The extended Hodgkin-Huxley Model

4.2.1.1 Neuron Cell

The data flow of the extended Hodgkin-Huxley model is illustrated in Figure 4.3. Before the calculations start, the parameters are initialized by some random functions, where the values are specified within defined ranges. Concurrently, the value of initial V_{Dend} (potential of dendrite) is given via control bus, reading from the configuration file. Due to parameters and V_{Dend} accessed frequently during the computations, both of them are stored in the local memory individually.

The inputs for a single neuron involve two parts; the potential from the neighbors, and the external stimuli. The cluster receives the inputs via the network and records them in the share memory at each time step first. Then, the neuron cells in the cluster can access the share memory to get their own inputs in the Round-Robin order.

The computation of Hodgkin-Huxley model can be split into two sub-computations. The first one can calculate the new V_{Axon} (potential of axon) and the states, based on the current V_{Dend} and parameters in the local memory. The other one, about generation of new V_{Dend}, should wait for the new V_{Axon} to continue with the calculation, together

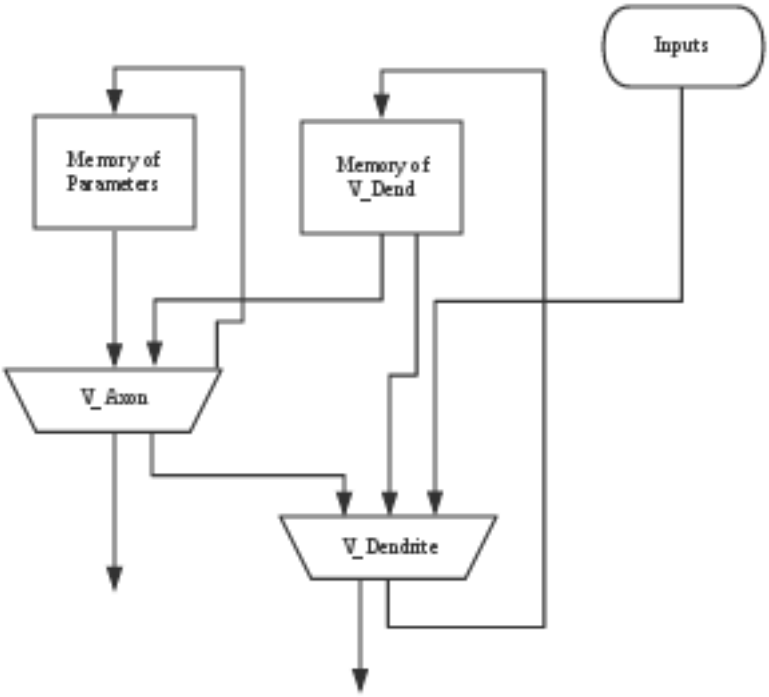


Figure 4.3: Data flow of a Hodgkin-Huxley model cell

with inputs and current V_{Dend}. After the calculation is performed, this new V_{Dend} will be transmitted to the neighbors, and the V_{Axon} is sent upstream directly into the network, which will be stored in the output files.

4.2.1.2 Physical Cell

Physical Cell (PhyC) calculates multiple time-shared neuron cells sequentially, due to the real brain time is slower than the clock frequency on FPGA [6]. The structure of PhyC is already introduced in Section 2.5 (Figure 2.5), and implementation of PhyC in the network can increase the amount of neurons simulated on FPGA.

4.2.1.3 Cluster

A single cluster in the system consists of several PhyCs, a share memory, and controller. A simple structure of cluster is shown in Figure 4.4.

In one cluster, several PhyCs are grouped and share one memory. At each step, the PhyC accesses this shared memory via a single memory port in the Round-Robin order. The shared memory is split into two sub-memories. All writes are done to one memories, while all reads are done from the other memory. After one system step, these two memories exchange their tasks.

The controller in the cluster is designed to implement several functions: i) The controller receives the packets and store them in the shared memory, according to the address contained in the packets. Meanwhile, the packets from PhyCs are sent upstream to the network by the controller. ii) When receiving a packet with id n_c, the controller will send a start signal to all PhyCs. On the other hand, when the controller received all calculation done signals from its PhyCs, it will issue a done packet (id=$n_c + 1$) to

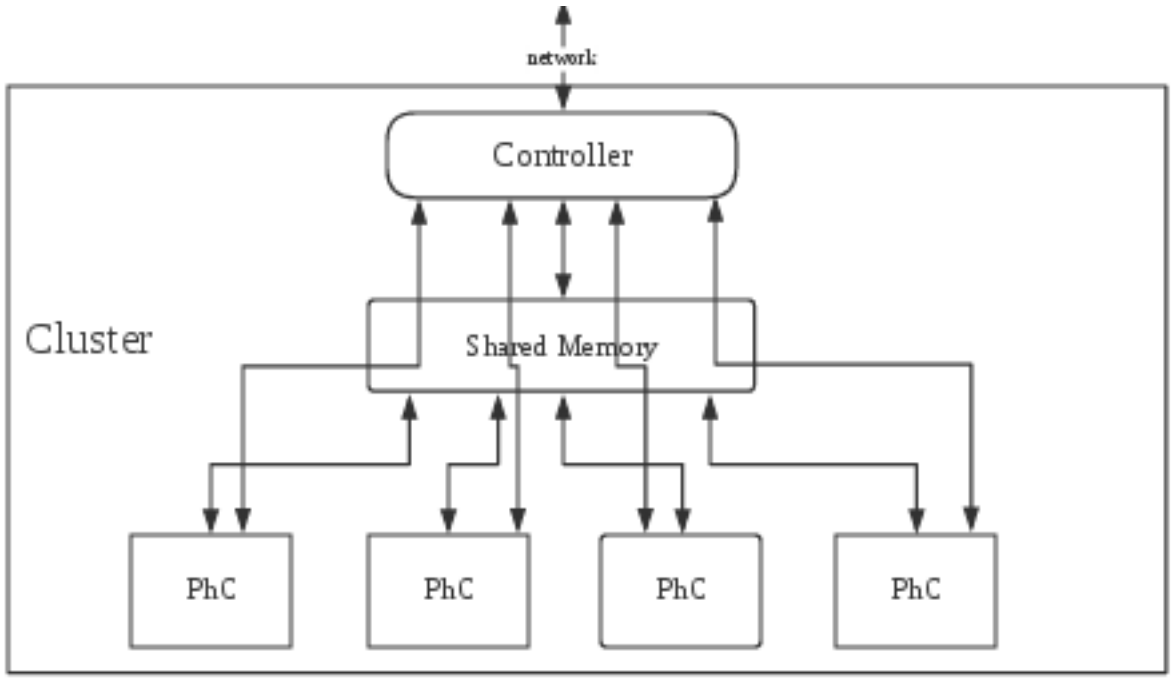


Figure 4.4: A simple structure of cluster [4]

the system controller. iii) The controller can check whether all required packets are received or sent out already at each step.

4.2.2 Integrate and Fire Model

In the Integrate and Fire model, the design of the cluster is almost the same as the one of the Hodgkin-Huxley model, except the implementations of the PhyC. A data flow of Integrate and Fire model is given in Figure 4.5.

The inputs are in the form of current, consisting of spikes from the neighbors, and the external stimuli. All the inputs are added together to derive a total current. This current is used in the potential function to calculate a new V (membrane potential), together with parameters and current membrane potential. Afterwards, the parameters are updated in the memory, and the new V is compared with a defined threshold. If V exceeds it, a new spike will be generated and transmitted to the neighbors via the network, meanwhile, the V and parameter u will be reset and stored in the local memory, respectively. On the contrary, if no spike is fired, the new V will be written back to its local memory, waiting for the next step.

```
1 V_dend_tmp = V_dend_buf[i]; // read the current potential of
                              // dendrite in the memory for the
                             //neuron i
  V_inf_tmp=E_L_tmp + I_tmp*R_m_tmp; // after the potential of
                                  //dendrite exponentially decaying,
                                 //the temporary potential V_inf_tmp is
                               //calculated. E_L_tmp, R_m_tmp are the
                              //parameters read from the local memory,
                             //I_tmp is the input current of neuron i.

6
```

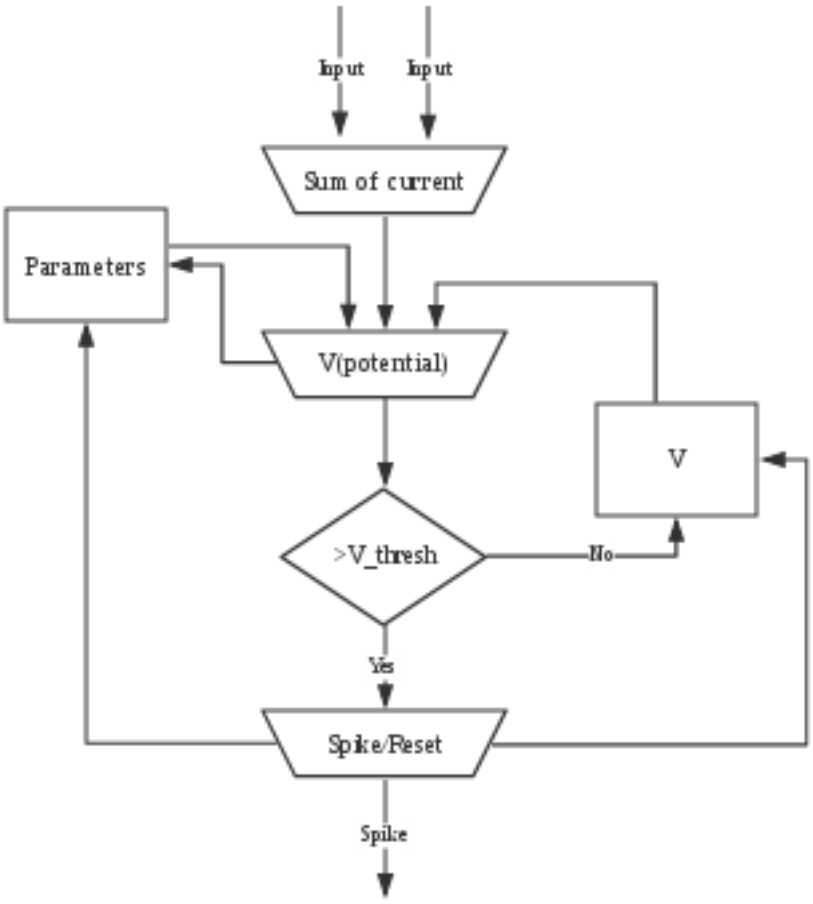


Figure 4.5: The data flow of Integrate and Fire model

```
V_dend_tmp = V_inf_tmp +(V_dend_tmp-V_inf_tmp)* exp_fun((0.0-dt_tmp)/
    tau_tmp); // the new current potential is calculated.

if(V_dend_tmp>-55.0)    // compared the potential with the threshold
{
  V_dend_tmp=V_reset_tmp;
  Axon_tmp=20.0; // neuron i fire a spike, the dendrite potential is
                 // reset.
 }else{
 Axon_tmp=V_dend_tmp;  // potential of axon
}

// Store the new potential of dendrite to local memory
slice_V_dend[i]=V_dend_tmp;
```

In general, the computation of Integrate and Fire model is the most simplest among three neuron models. Consequently, it is applicable for simulation of large amount of neuron cells in the network, if the biophysically meaningful features of the neuron cell is not an issue.

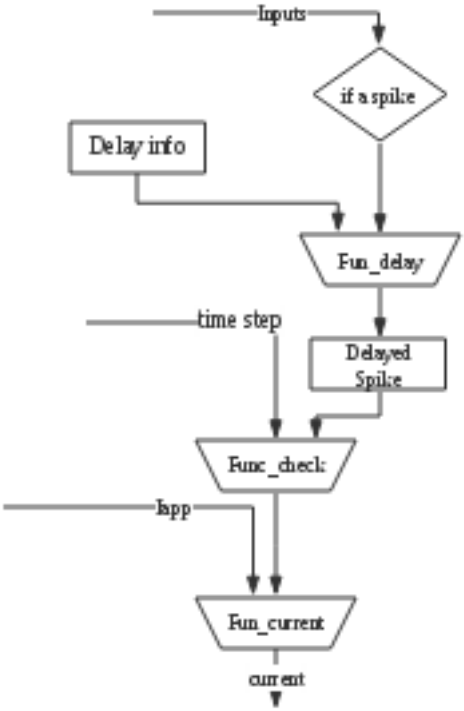


Figure 4.6: The data flow of the delay implementation

4.2.3 Izhikevich Model

The Izhikevich model is usually used to investigate the patterns of the spike train. Additionally, in this system, provisions are made to dynamically changes synaptic weights. The whole design of Izhikevich model can be sub-divided as three parts: spike generation, delays, and STDP.

4.2.3.1 Axonal Conduction Delay

The implementation of axonal conduction delay includes three basic functions; f_{delay} (defines the exact delay time for each arriving spike), f_{check}(check whether spikes in delay buffer are compatible with simulation time step), and $f_{current}$ (generates the current).

At each step, all arriving inputs from neighbors should be compared first. Only the input, whose potential is larger than the defined threshold, is recognized as a spike from neighbor. The delay information is stored in a local memory, which record the exact delay time for each neighbor. If the spikes are accepted, the f_{delay} accompany these spikes with delay information, and then store them in a delay buffer. Next, according to the current system simulation time, f_{check} checks whether there are some spikes can be fetched out of the buffer. Afterwards, the total input current can be calculated in the f_{check}, together with the external signal. The basic data flow is presented in Figure 4.6 .

In the designed network, the generation, and transmission of packets are performed in the same cycle. Consequently, to implement the delay between neurons, an additional spike delay buffer is placed in the module, which enables that the spikes can be received at the expected time. The thesis assumes that the currents of spikes from neighbors are depending on the pre-synapse weights, thus, the delayed spike buffer only need to record the actual arriving time for each pre-synapse neighbor. When the spikes are picked from buffer, the corresponding input currents are calculated based on their

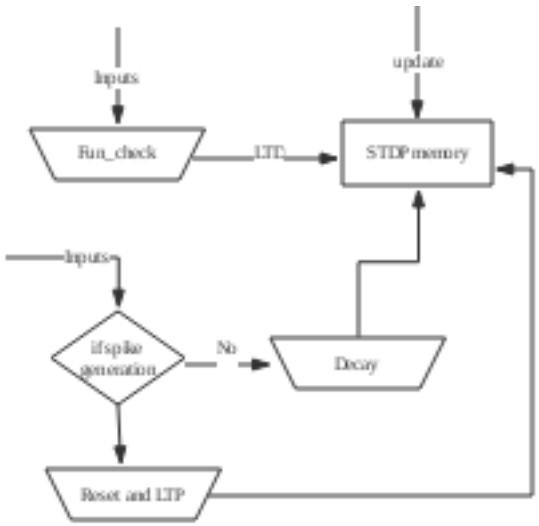


Figure 4.7: Basic data flow of STDP in the system(rename the functions)

synaptic weights. The basic delay implementation in SystemC can be showed as below:

```
  if(v_tmp==35)  // check whether the receiving signal is a spike
2  {

D=(*iter).delay[neighbours.ID]+t-1;  // calculate the actual time step,
    // according to delay information. t is the current system time step.

(*iter).fire[D].neighbours[neighbours.ID]=1;   // Store the spike in
    // delay buffer. At time step D, spike of neighbor.ID can be fetched.
7
}

  for(int i=0;i<config.getCellCount();i++)
   {
12  if((*iter).fire[t].neighbours[i]>0) // whether spike from neuron i can
     //  be fetched at time t
   {

   slice[(*iter).Infoli_ID].I_c +=Sweight[i].s[(*iter).ID];
     // with synaptic weight, the current of spike i is calculated. Then
        // it is added to the total input current.
17
   }
```

4.2.3.2 STDP

The implementation of STDP refers to several important variables, *e.g.* long term potentiation (LTP), long term depression (LTD), pre-synapse weight, pre-synapse derivatives, post-synapse weight, and post-synapse derivatives. A global memory is employed for the general learning network, to perform the dynamic change of parameters of STDP (Figure 4.7).

When a single neuron of `Izhikevich model` receives the spike from a pre-synapse neighbor, the pre-synapse derivative will be depressed by the variable LTD. Next, it updates the new value of STDP parameter in the global memory, so that the neuron firing a spike can know the change of its post-synapse derivative immediately (the pre-

synapse weight/derivatives of former equals to the post-synapse weight/derivatives of latter). In the case that a spike is generated by the neuron itself, the variables LTD and LTP will be reset. Furthermore, all the post-synaptic weights of spiking neurons, will be increased. These changes are also written into the global memory, so that the post-synapse neighbors can share them. On the other hand, if no spikes generated, all six variables will decay according to STDP rule. The example of STDP implementation in SystemC is below:

```
if(slice[i].V_dend>=30) // check whether neuron i fire a spike
{

 Sweight[slice[i].ID].LTD=0.12;
 Sweight[slice[i].ID].LTP[t+config.Max_Delay]=0.1;//reset LTP and LTD

 for(int k=0;k<config.getCellCount();++k)
 {
 if(neighbour_list[k].slices[slice[i].Infoli_ID]){
    //check whether the cell k is a neighbor of neuron i.
 Sweight[k].sd[slice[i].ID]+=Sweight[k].LTP[t+config.Max_Delay-slice[i
    ].delay[k]-1]; // update the synaptic weights of neuron k

 }
}
 Sweight[slice[i].ID].LTD=0.95*Sweight[slice[i].ID].LTD;
 Sweight[slice[i].ID].LTP[t+config.Max_Delay+1]=0.95*Sweight[slice[i].
    ID].LTP[t+config.Max_Delay];    //if neuron i not fire a spike,
    // the parameters decay, according to STDP rule.
```

4.2.3.3 Spike Generation

The overview of data-flow implementing the Izhikevich model is illustrated in Figure 4.8.The details of delays and STDP are described in the previous two sections. In this designed module, the total current is calculated first, depending on the external inputs, pre-synaptic weight, and delays. Then, f_V generates the new membrane potential based on the calculated current, current V_{dend}, and the parameters. Both the new potential and parameters are written to the local memory. Afterwards, this new membrane potential is compared with defined threshold, and determine whether a new spike can fire. No spike generation means that the parameters of STDP should be decayed proportionally by the f_{decay}. Otherwise, the membrane potential and some specific parameters are reset to the pre-defined values, and fed back to the corresponding local memories. Concurrently, all the post-synaptic weights of spiking neurons are increased. The spike generation in SystemC can be described as below:

```
slice[i].V_dend=slice[i].V_dend+slice[i].tau*(0.04*slice[i].V_dend*
    slice[i].V_dend+5*slice[i].V_dend+140-slice[i].u+slice[i].I_c);
// calculating the potential of dendrite for neuron i

slice[i].u +=slice[i].tau*slice[i].a*(slice[i].b*slice[i].V_dend-slice
    [i].u);
// update the parameter u in the memory
```

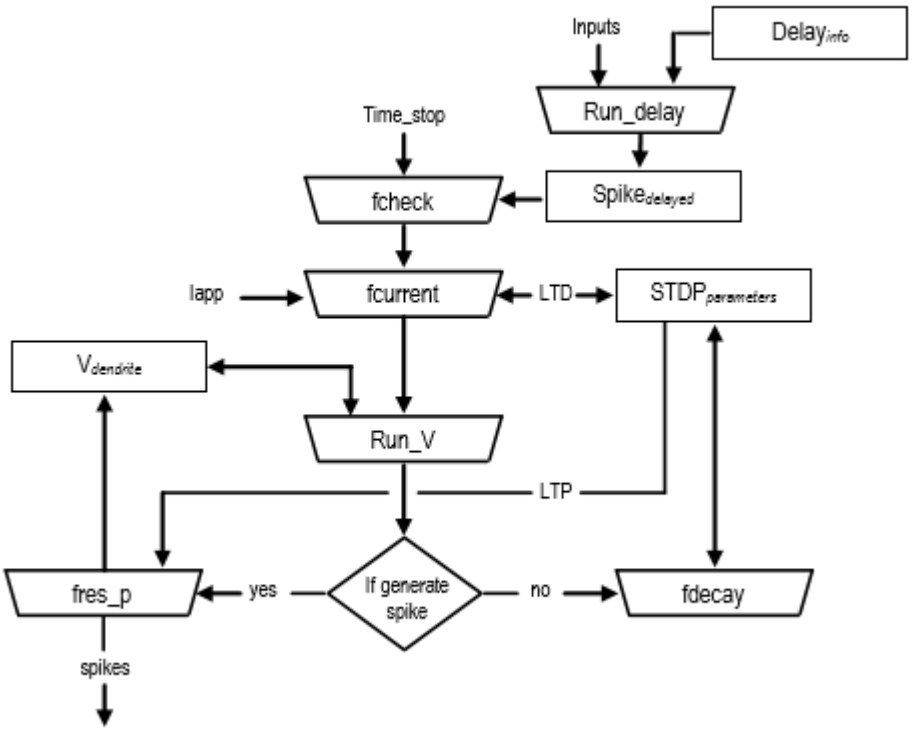


Figure 4.8: The date flow of Izhikevich model (2006)

```
    if(slice[i].V_dend>=30) // whether neuron i generate a spike
     {
9      slice[i].V_dend=-65;  // reset potential of dendrite
       slice[i].u +=slice[i].d; // update the parameter u
       slice[i].V_axon=35;  // reset potential of axon
     } else
     { slice[i].V_axon=slice[i].V_dend; } // no spike generated
```

4.3 High-level Synthesis

When implementing the designed network on the FPGA, the resource requirements and time constraints must be taken into consideration. To insure that the synthesis of the system fulfills the requirements as well as FPGA implementations, several adjustments are made in the system.

4.3.1 Optimization with directives

Using Vivado HLS to synthesis the system , various settings can be conveniently defined by the users. Some main directives used in the system are listed below.

- During the implementation of three neuron models, calculation of divider is the least amount of times used, which consumes considerable hardware resources. Consequently, the number of dividers is limited in the SystemC code with:

```
#pragma HLS ALLOCATION instances=FDiv limit=1 core
#pragma HLS ALLOCATION instances=DDiv limit=1 core
```

- An appreciate placement of arrays in the memories can effectively reduce the hardware cost. According to the requirements, some arrays (e.g. array storing potential of dendrite) are allocated the local memories individually, which need to be operated concurrently. Conversely, other arrays (i.e.,array on the state parameters) can share the same local memory. (neurons in the same PhyC perform their calculations sequentially)

```
#pragma HLS RESOURCE variable=slice_V_dend core=RAM_1P_BRAM
#pragma HLS RESOURCE variable=slice_delay core=RAM_1P_BRAM
#pragma HLS RESOURCE variable=spike_delay core=RAM_1P_BRAM
```

- Several wait() statements are inserted into the code to notify the scheduler of the arriving of signals outside the scope. *e.g.* the PhyCs access the shared memory in the cluster in the Round-Robin order. By instructing wait statements, scheduler can schedule the access to memory for specific PhyC at each cycle, to ensure that no stalls occur and that each one can read or write the shared memory in time.
- The calculations of neuron models support both double and single floating point precision.

4.3.2 Adjustments of System for HLS

4.3.2.1 Hodgkin-Huxley model

The implementation of the extended `Hodgkin-Huxley model` includes multiple computations of exponential functions. If the neuron cells directly make use of the units of exponential function to perform the computation, the resource requirement will increase significantly with the growth of the amount of neurons simulated. Consequently, a special component of exponential calculation is designed [4], which can be shared by the PhyCs in the cluster. The number of these components is not just limited by only one in the cluster. Instead, it is up to the integer value of PhyCs/ExpCs (PhyCs stands of the number of PhyCs in one cluster, and ExpCs means the maximum number of PhyCs supported by one exponential component.) Both two values can be varied in the configuration part.

As described in the previous section, the computations of `Hodgkin-Huxley model` can be divided into two sub-calculations. The first one is the axon potential and state parameters computations, which includes multiple exponential calculations. The inputs to this part are named low priority inputs, and are sent to multiple ports of exponent core. The second sub-calculation, the generation of dendrite potential only need one exponential operation. The exponential operand is read by a single port, which is shared with multiple PhyCs (Figure 4.9).

When the exponential operands arrive at the ExpC, the operations are performed in pipeline. A shift counter is a FIFO buffer and placed to record the input address.

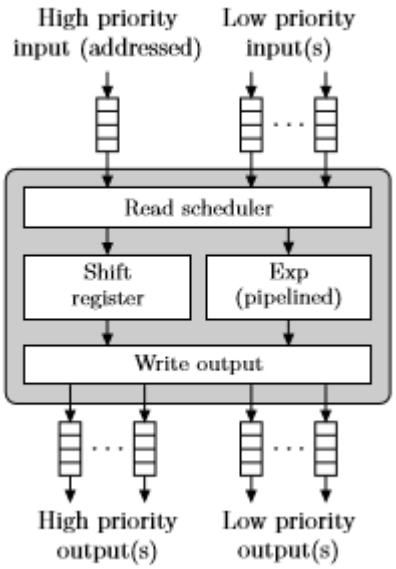


Figure 4.9: Structure of exponent core [4]

Once the new input is read, its address will be inserted to the bottom of buffer. While an exponential results is generated at each cycle, the address on the top of the counter will be fetched. Thus, this data can be written back to the specific PhyC via the output buffer. At last, all the entries in the counter shift up one place, and then the bottom of entry is available to the next input.

4.3.2.2 Integrate and Fire Model

Unlike the `Hodgkin-Huxley model`, the `Integrate and Fire model` only needs one exponential operation in the computations. Consequently, the design of this model includes two options: either using the exponential component as described above, or directly performing the operation on the unit of exponential function. In this thesis, both types of implementations are developed. After comparing the performance using Vivad HLS, the result shows that the design with ExpC in this model costs about 2.5 times more hardware resource than the other one does. As a result, the design of later is adopted in this system.

4.3.2.3 Izhikevich Model

The computations of `Izhikevich model` does not include any exponential operations (consequently, approximately 10% percents of memory is saved). The STDP parameters are localized in each PhyC. In comparison with the network saving STDP parameters in the global memory, this adjustment reduces the memory latency.

The main problem encountered when implementing this model, is how to inform the neighbors of spiking neuron when the spike is fired. This means that the synaptic weight between two neurons should be kept consistent, and potentiation or depression of synaptic weights needs to be scheduled to occur at the right time.

After optimizing the design of this part, an extra connectivity matrix is introduced in the system. In comparison with original matrix recording the connections from the pre-synapse neighbor (e.g. a_{ij} in the matrix means that node a_i connects to the a_j), the new matrix stores the information of post-synapse neuron. Based on [4], the way

of creating connectivity matrix among neurons is quite simple and fixed. Each neuron cell is both the pre-synapse and post-synapse neighbor for the connecting node (e.g. if a_i connects to the node a_j, then there must be an inverse connection from the node a_j to a_i). An optimized design is extended to suit for the more flexible method of creating the connectivity, which can be developed in the future. After both matrices are defined, their values are sent to the cluster memory in the form of packets, with the type 4 or 5, (see Table 4.1), respectively.

type 1	Dendrite potential
type 2	Axon potential
type 3	Iapp
type 4	Matrix from pre-neuron
type 5	Matrix to post-neuron

Table 4.1: The different input types

At each step, all the dendrite potentials are updated in the local memories for the neurons. When a new cycle of calculation starts, the cluster reads the required data in the shared memory and transmit them to the corresponding neurons via FIFO buffers. The potentials from pre-synapse nodes are sent first. After the cell receives a packet with special value (e.g. the data is less than zero), it is informed that the following packets are from the post-synapse neurons.

In the module of Izhikevich neuron, a counter is placed to distinguish type 4 and 5 potentials. After starting to receive the potentials from neighbors, if one input exceeds the pre-defined threshold, it means that the corresponding pre- or post-synapse neighbor fires a spike. Thus, the pre- or post-synapse weight should be increased or decreased at some time later (this depends on the delay information for this pair of nodes), which are updated in the local memory.

In a neuron itself, if the new membrane potential is smaller than the threshold, the synaptic weights will decay according to STDP rule. On the other hand, if a spike is fired, the synaptic weights for all the neighbors are enhanced or reduced at once. Subsequently, the new membrane potential will be transmitted to both pre- and post-synapse neighbors via network.(the design before adjustments only sends the potential to the post-neurons).

In general, each neuron stores the individual STDP parameters in its own local memory. By receiving the packets from both sides of neighbors and detecting the generation of fire itself, the dynamic change of STDP can be implemented in the system (Figure 4.10).

4.4 Summary

This chapter presents the implementation details of system design which is described in the chapter 3. It mainly introduces the improvement based on the [15], to implement three neuron models in the real-time, data-flow, learning network. Due to the resource and time constraint, some adjustments are made in the design to meet the high level

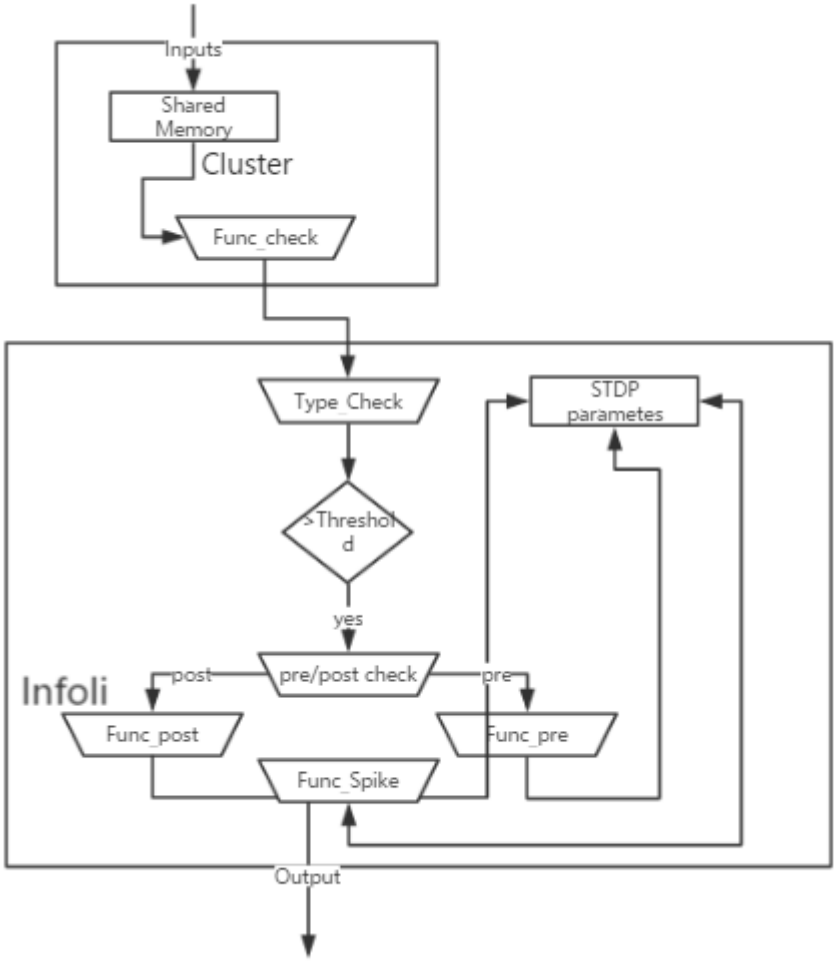


Figure 4.10: Date flow of STDP design after adjustment

synthesis, so that the system can be implemented on the FPGA . The following chapter presents the performance evaluation with different system configurations, and the optimal configurations are explored. To the end, the impact on varying the buffer size is explored.

5 Performance Evaluation

In this chapter, multiple test cases are designed with different configurations. The results include accuracy, property, latency, and resource usage, etc, and consequently, the optimal configurations for the three neuron models are defined. In addition, the change of buffer depth in the designed system is explored, since it can have potential effect on the implementation.

5.1 Evaluation method

Within this section, the details of evaluation method are discussed. By changing the configuration files, multiple neuron cells logic set-ups can be created. After the inputs are defined, the system can generate and store the results in the DAC files. These results can be used to verify the correct behaviour of ION, compared to the reference models [29, 41, 4]. Subsequently, the corresponding properties [27] of neuron cells can be plotted in the form of graphs with the outputs. In order to calculate the latency for each test case, a SystemC trace file is added in the system. By tracing some specific signals, the latency can be calculated with the help of the tool GTKWave [9]. Finally, after a test case is simulated and synthesized by Xilinx Vivado HLS, the resource usage reports can be found.

In this thesis, the best case means the maximal number of neuron cells that can be simulated in the designed network within the required resource and latency limitation.

5.1.1 Simulation configuration

In the system, two configuration files are defined to generate the specific test cases. The one is used to define the parameters of system simulation, and the other one is set to describe the properties of neuron cells. (1) The example of the former can be showed as below

```
export DIM1='1'# #PhyC/Cluster
export DIM2='25'# TSF
export DIM3='2'# #Clusters

export DO_SIM=FALSE
export DO_SYNTH=TRUE
export DO_COSIM=FALSE
export DO_IMPL=TRUE

export TARGET_PART= xc7vx550tffg1927-2
export PRECISION=SINGLE # FLOAT | DOUBLE
```

```
export SIM_T=1000000 # Simulation time (ns)
export SIM_STEPS_T=120 # Simulation step #
export START_PERIOD_T=60000 # Start pulse period
```

This configuration file can be split into four parts. i) The first part refers to the size of the ION. Dimension one is on how many PhyCs are in one cluster, and dimension two means the time-share-factor (the number of neuron cells in one PhyC). The last dimension is used to determine the number of clusters in the system, as well as the size of routing network. According to these three parameters, the total amount of neuron cells for one test set equals to:

$$Neurons_{total} = DIM1 \times DIM2 \times DIM3 \tag{5.1}$$

ii) The second part relates to the parameters on the steps where the synthesis run with the tool of Vivado HLS. Once the parameters are set to true, the Vivado HLS will implement the corresponding operations. The procedure of synthesis stops at the step where the Boolean value of parameter is false.

iii)The third part is used to select the FPGA to implement the test case, and determine if the calculations are performed with single or double float precision. In this thesis, all the test set-ups are operated on the FPGA board xc7vx550tffg1927-2.

iv) The last part informs the system how the simulation progresses. The simulation time is defined to avoid the endless of operation, in case the deadlock or some other errors occur. The simulation steps determine how many times each neuron cell should perform the calculation in this case. At each time step, the corresponding results are stored in the DAC file, which can be used to generate the graphs of properties and compare the accuracy with the reference model. The final parameter describes when the pulse starts in each cycle.

(2)The second configuration file specifies the types of neuron cells simulated in the system, due to the numbers of parameters in the neuron models are different. Furthermore, in one model, one set of parameters can determine a specific property of neuron[28]. Once the parameters are defined, the corresponding property can be expressed as a specific spike train. e.g. one configuration file of `Izhikevich model` can be written as below:

```
define V_DEND_ -65.0f
define b_ 0.2f
define c_    -65.0f
define d_ 0.01f
define u_ -14.0f
define I_C_ 0.5f
define Tau_ 0.25f
define V_SOMA_ -65.0f
```

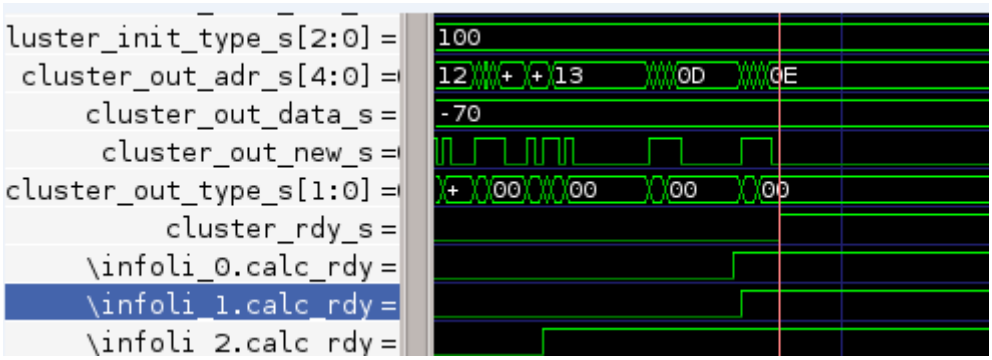


Figure 5.1: An example of wave diagram

5.1.2 Design simulation

After a test case is configured, the system can be simulated, and the results of selected neuron cells are stored in the output file (DAC) at each time step. The results are compared with the ones from reference model [29, 41, 4], to examine the accuracy. Concurrently, with the number of steps and the corresponding results, the tool of Matlab can generate one of the twenty properties of neuron cells [27]. By changing the sets of parameters, `Integrate and fire model` can exhibit 3 properties, while the `Hodgkin-huxley model` and `Izhikevich model` can perform 19 and 20 different spike trains (Figure 2.3), respectively.

As introduced in the Chapter 3 and Chapter 4, the cycle time is smaller than the real time speed of brain. Consequently, multiple neuron cells can be grouped together as one PhyC to perform the calculations sequentially at each step, as long as the total time latency does not exceed the threshold. In order to inspect the latency, the signal trace file is added in the system. By tracing some specific signals, the relative information can be stored in the trace file after the simulation is completed. The setting of signal trace can be defined as below:

```
sc_trace (fp,top.cluster_in_str,"cluster_in_str_s");
sc_trace (fp, top.cluster_in_ack,"cluster_in_ack_s");
sc_trace (fp,top.cluster_out_type,cluster_out_type_s");
sc_trace (fp, top.cluster_out_data,"cluster_out_data_s");
sc_trace (fp, top.cluster_out_new,"cluster_out_new_s");
```

With the help of GTKWave, the signal timing information can be inspected in the form of digital wave. Figure 5.1 gives out an example of simulation. At each time step, the neuron cells in one PhyC perform their own calculations in pipeline (the high pulse of signal calc means that the corresponding action is completed). After all the required data is sent to the neighbours via the routing network, the signal $cluster_ready$ goes high. Thus, the final latency of PhyC can be derived as:

$$Latency = Time1 - Time2 \tag{5.2}$$

where the Time1 stands for the arriving time of starting signal, and Time2 is the time when the cluster is ready to receive the new signal.

Clock	Target	Estimated	Uncertainty
default	10.00	8.46	1.25

Figure 5.2: An example of timing performance analysis

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	-	-
FIFO	0	-	53	251
Instance	16	16	4409	6395
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	1	-
Total	16	16	4463	6646
Available	2360	2880	692800	346400
Utilization (%)	~0	~0	~0	1

Figure 5.3: An example of hardware utilization report

5.1.3 Design synthesis

If a configured logic set-up can be synthesized successfully by the Vivado HLS, some resource usage files will be created. The report named cell-top-cssynth records the whole utilization of hardware and timing performance for the simulated test case. Figure 5.2 shows the summary of timing performance in the report. In the Figure 5.3, the resource usage, including BRAM, DSP, FF, and LUT, are shown. The utilization of each resource should not exceed the available amount, otherwise the corresponding test case cannot be really implemented in the corresponding FPGA.

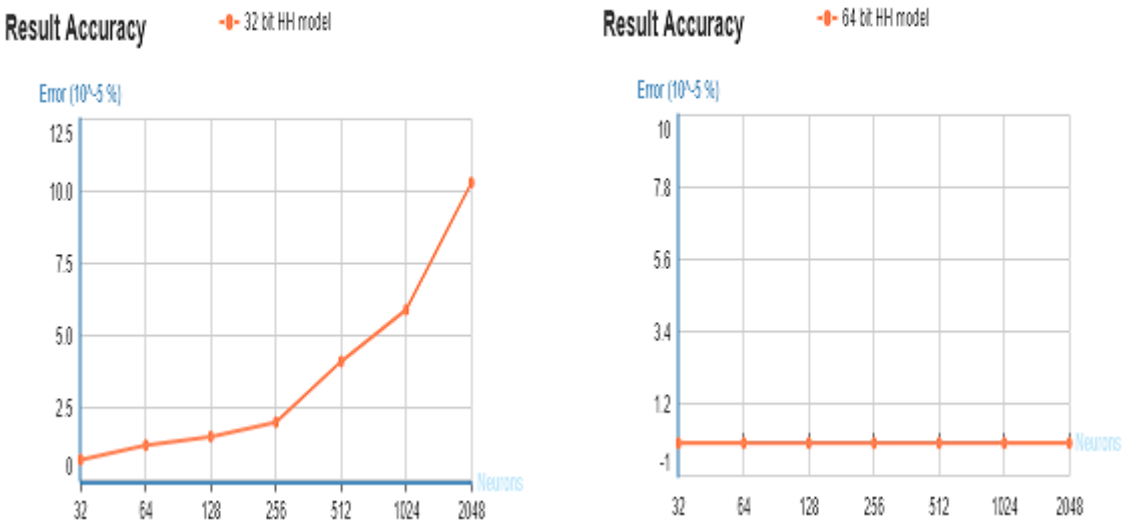
5.2 Evaluation Results

In the section, the evaluated results from multiple test cases are discussed. At the beginning, the result accuracy and properties gained for the simulated neuron models, compared with the reference models [29, 41, 4]. Secondly, the relationship between latencies and different ION logic set-ups are analyzed. Afterwards, the designed network is synthesized with Vivado HLS, to obtain the reports of hardware utilization. At the end, the buffer depth is discussed to examine if the size of the buffer can influence the accuracy.

5.2.1 Accuracy result

Besides the mistakes caused by packet loss, the system precision can also influence the accuracy, especially when the size of network or system steps is large. In this thesis,

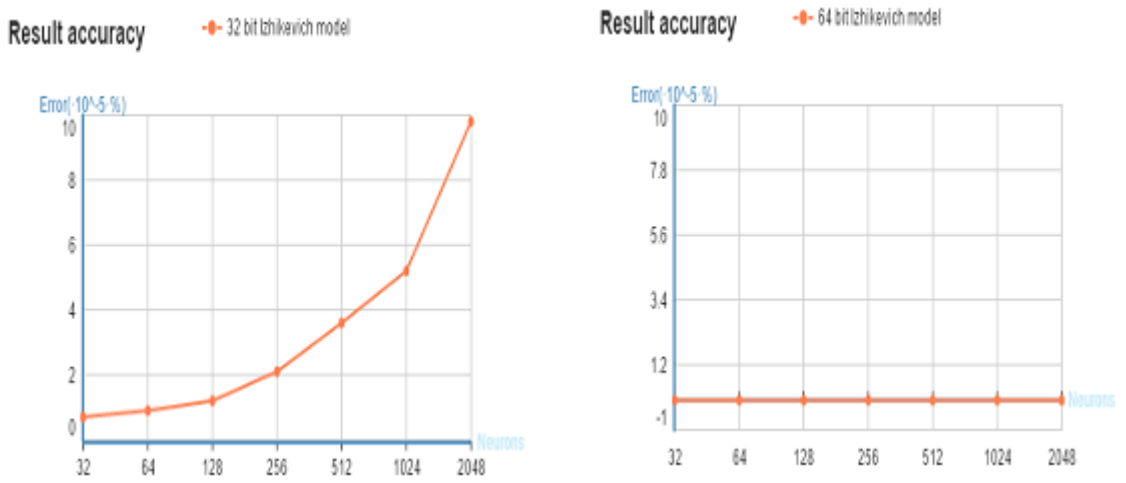
the original system in SystemC is designed as 16 bit fixed point precision. To meet the requirement of high-level synthesis, some optimization is made. The adjusted system is 32 bit or 64 bit floating point precision, which can be switched in the configuration file. Once the results are gained, these can be compared with the reference results [29, 41, 4], to get the differences (accuracy).



(a) 32-bit Hodgkin-Huxley model

(b) 64-bit Hodgkin-Huxley model

Figure 5.4: Maximal 32-bit/64-bit errors compared to the reference results [4], for the Hodgkin-Huxley model, implemented in the adjusted network

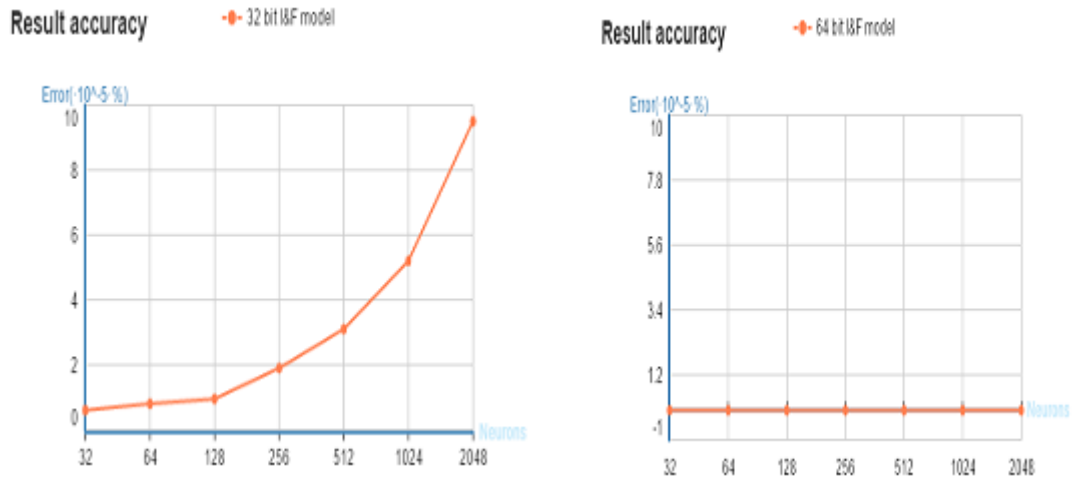


(a) 32-bit Izhikevich model

(b) 64-bit Izhikevich model

Figure 5.5: Maximal 32-bit/64-bit errors compared to the reference Matlab results [29], for the Izhikevich model, implemented in the adjusted network.

The results indicate that there are no differences with the reference ones for the three neuron models in the original system. However, if the configuration of test case



(a) 32-bit Integrate and Fire model

(b) 64-bit Integrate and Fire model

Figure 5.6: Maximal 32-bit/64-bit errors compared to the reference Matlab results [41], for the Integrate and Fire model, implemented in the adjusted network.

are beyond some limitations (i.e., the amount of PhyCs should smaller than 13), the simulated neurons can not generate the correct results. The reason is that the bandwidth of network is limited. If too many packets are transmitted concurrently, the packets loss likely occurs in the network.

The situation is quite same, while multiple cases of neuron models are tested in the adjusted system. Within the appreciate configurations, the results of 32-bit design have high accuracy and the 64-bit errors are 0, compared with the reference models (Figure 5.4, Figure 5.5 and Figure 5.6).

5.2.2 Properties

Besides the high accuracy, the simulated neuron models are also expected to output the correct response of biological behaviours in the designed network (Figure 5.7 and Figure 5.8). The property of neuron can be defined by setting the specific values to parameters. Hence, when a suitable pattern of inputs is given, the outputs can be abstracted as a spikes train by the plot tool. These properties have different important roles in the brain. *e.g.* the neurons with tonic bursting contributes to the gamma-frequency oscillations in the brain, and tonic spiking can be observed in cortical neurons [27].

5.2.3 Simulation time

In order to compare the simulation time among the neuron models, the same configured logic set-ups are defined for each of them. The command time is employed before running the systems. After the simulation step is completed, the total time consumption can be gained.

Figure 5.9 and 5.10a illustrate the simulation time on different sizes of networks in the original system. The simulation time of three models increases significantly when the network size is beyond certain range (about 1000). Given the same configurations in

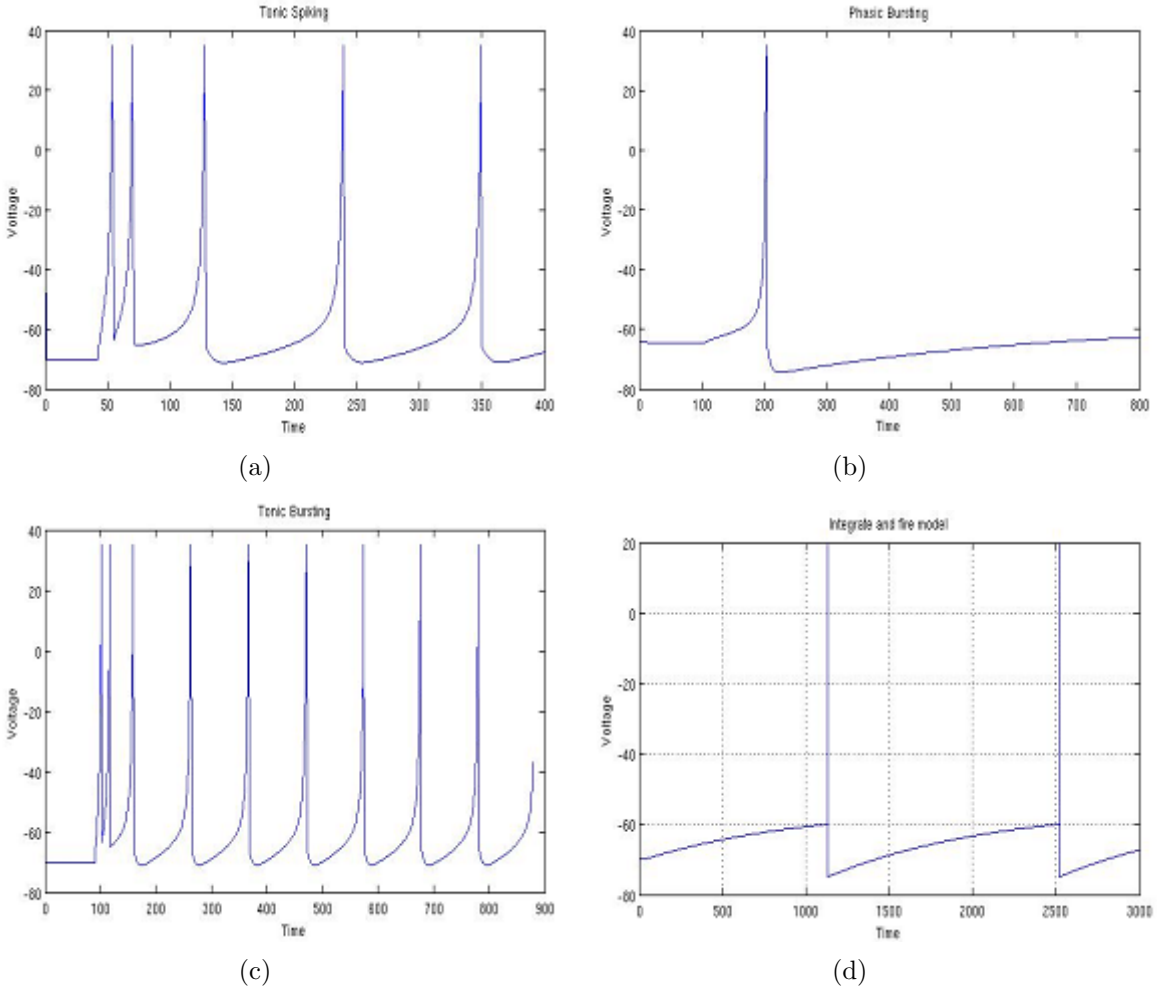


Figure 5.7: The examples of properties generated by the neurons in the designed network base on [15]

the network, Integrate and Fire model always costs less time than the other two model does, which is proportional to the complexity of models. In addition, if the configuration is beyond some upper bounds which can guarantee the high results accuracy, the growth of time will become slower, even minus, due to the packets loss. The problem can be partially fixed by enlarging the buffer depth. *i.e.,* the simulation time with 4096 cells of Integrate and Fire model, is 33.64s. After double the buffer size of routers in the network, the packet loss is solved, and the time grows to 44.65s.

Figure 5.10b and 5.11 show the simulation time on different network size gained in the adjusted system. The results reveal that the behaviors of three neuron models are quite similar to the ones described above. In addition, simulation time cost by the two designed network are compared within the same configurations. The results indicate that the original system run faster. The reason is that, to implement the design on the FPGA, the system simplifies routing mechanism in the routers, which causes the extra growth of communication time in the network.

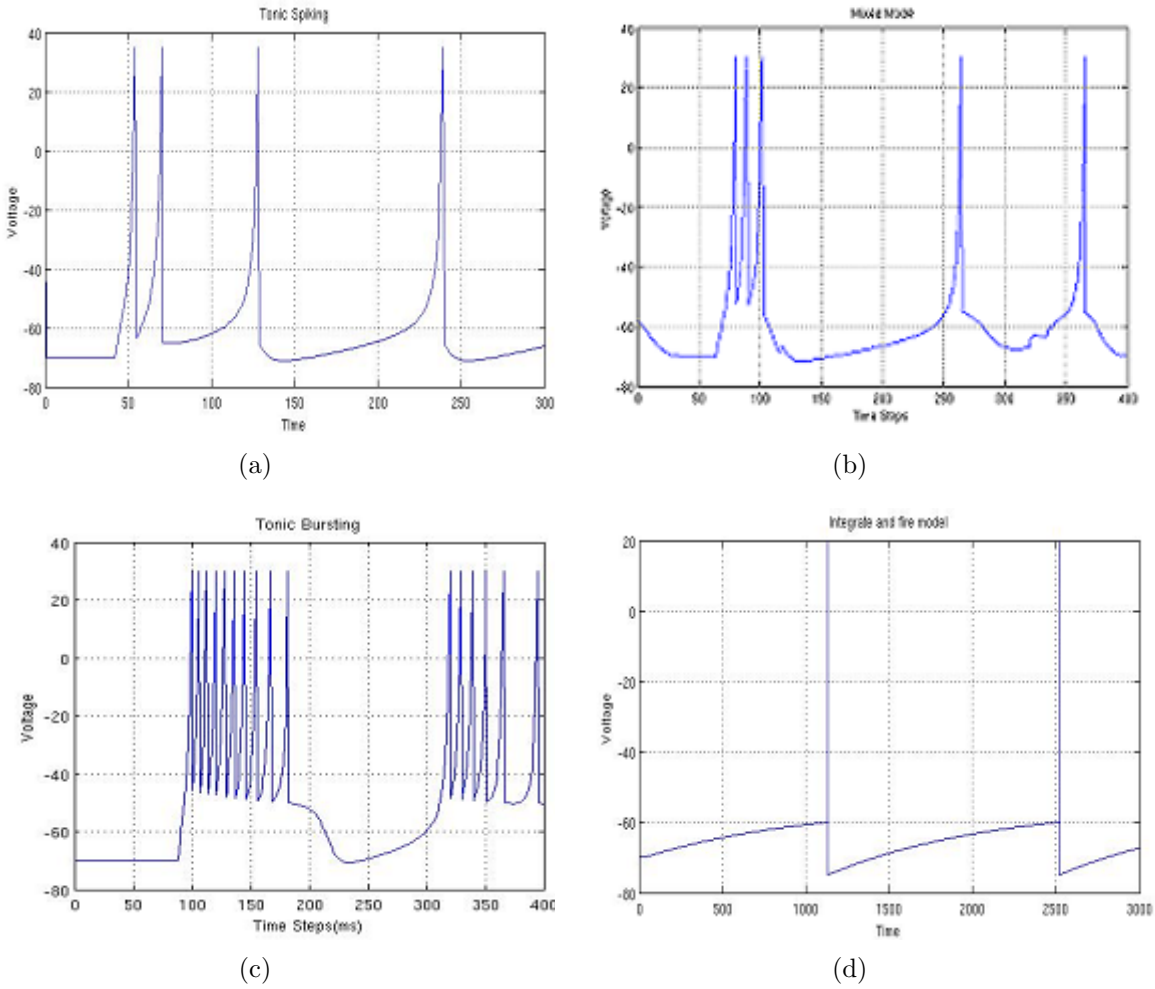
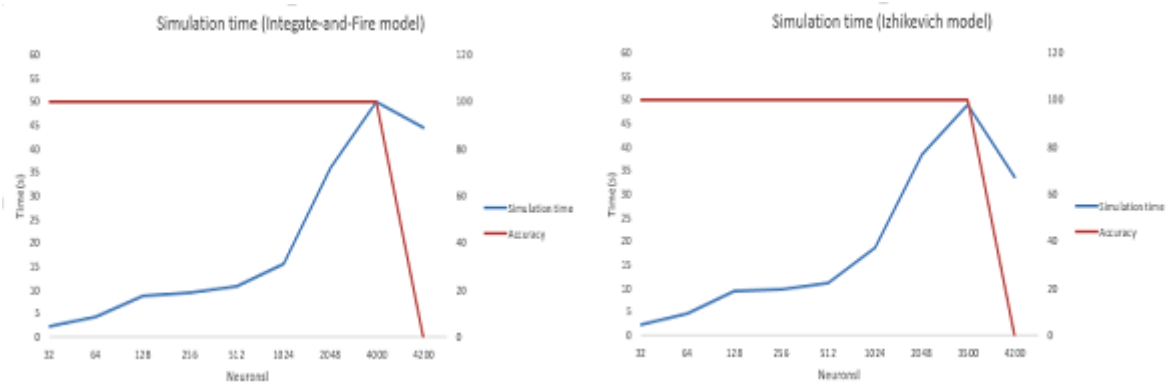


Figure 5.8: The examples of properties generated by the neurons in the adjusted network for the synthesis, based on [4]

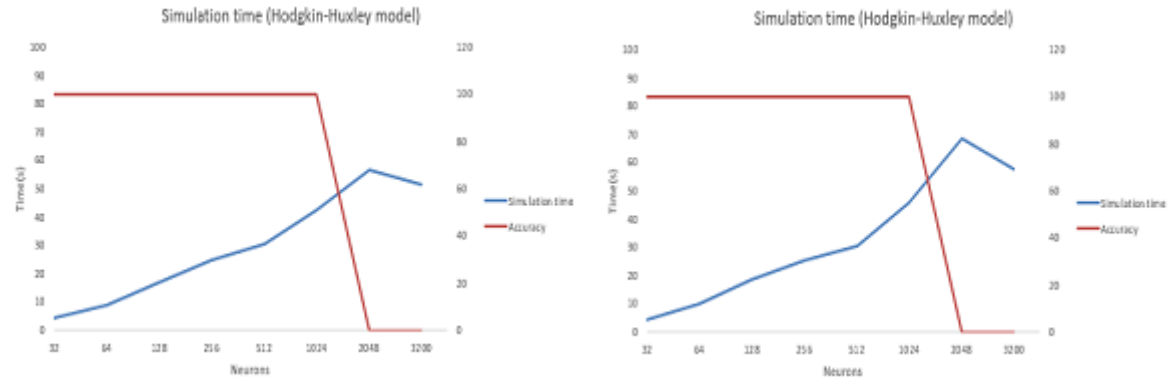
At the end, the relationship between the configuration of network and the simulation time is discussed. According to the equation 5.2, the size of a test case is determined by three dimensions, TSF, PhyC, and cluster. Given the same amount of simulated neurons, the influence of changing the dimension can be concluded as below:

- While increasing the number of clusters, the network size is also enlarged, as more routers are created in the network. Thus, the growth of network size leads to more communication time among clusters.
- Time-Share-Factor (TSF) means that multiple cells can be grouped together to operate sequentially at one time step. Thus, the larger TSFs are defined, the less total simulation time is required. However, the value of TSF should be limited, in case that the total latency of time-shared cells exceeds the threshold, which is the actually biological brain time [6].
- The dimension of PhyC equals to the number of PhyCs working in one cluster. As multiple PhyCs share one memory in the cluster and access it in Round-Robin order, the increase of PhyCs also causes more simulation time. After analyzing different test cases, the maximal number of PhyC is found to be 13. If the value is larger than 13, segment error will be issued during the simulation.



(a) Integrate and Fire model (b) Izhikevich model

Figure 5.9: The simulation time gained in the original network, for Integrate and Fire model (left), Izhikevich model (right).



(a) Hodgkin-Huxley model (b) Hodgkin-Huxley model

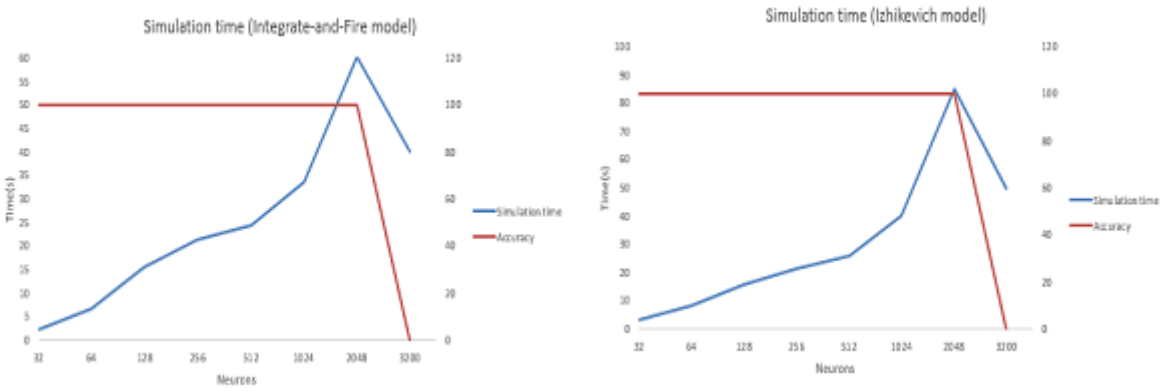
Figure 5.10: The simulation time of Hodgkin-Huxley model gained in the original network (left), adjusted network (right).

5.2.4 Latency results

The following parts will focus on the analysis of experimental results from the adjusted system which is design to be implemented on the FPGA, including latency, resource usage, etc. In the Section 5.1.2, it describes how the latency of PhyCs are calculated in the network. The latency is dependent on the logic set-up defined by the configuration file and the size of neuron network. More neurons in the network would absolutely require extra time to complete the calculations. Moreover, given the same size, the varying dimension can influence the latency of PhyCs.

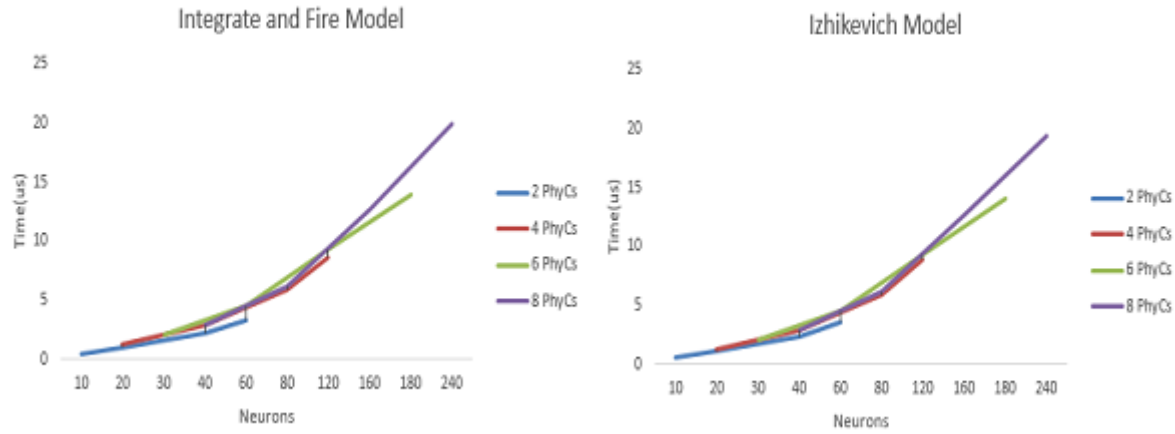
Firstly, the relationship between the latency and TSF is explored in Figure 5.12 and 5.13a. In these test cases, all PhyCs are connected to a single cluster. By increasing the TSFs, the latency of designed neuron models have a nearly linear growth in the learning network. This effect is implemented by introducing more computational hardware to the neuron module. In addition, given the same logic set-ups for the three models, the Integrate and Fire model has a little smaller latency than the others. Consequently, the latency of neuron is also dependent on the complexity of computations.

Next, the effects on varying PhyCs in the network are presented, where the number of cluster controller is only one, and the TSFs are also pre-defined. Through the Figure



(a) Integrate and Fire model (b) Izhikevich model

Figure 5.11: The simulation time gained in the adjusted network, for Integrate and Fire model (left), Izhikevich model (right).



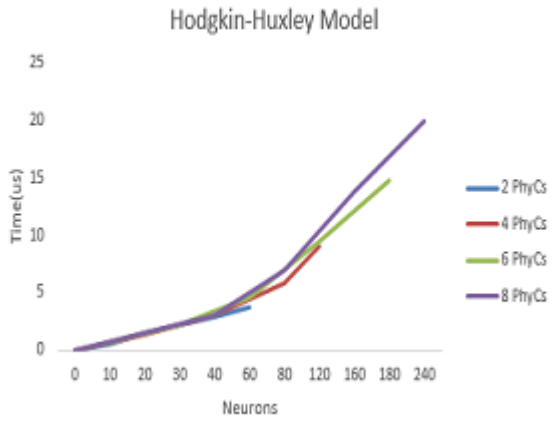
(a) Integrate and Fire model (b) Izhikevich model

Figure 5.12: The effect on latency by varying TSFs in the neuron network.

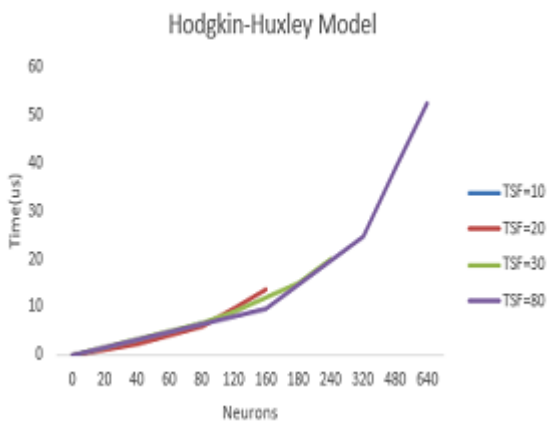
5.14 and 5.13a, some conclusions are drawn: i) The latency of neuron models still have the nearly linear effects in the design, while increasing the number of PhyCs. ii) Due to computational complexity, the Integrate and Fire model requires the smallest latency among three models, within the same configurations. iii) when the size of network is specified, increasing the amount of PhCs costs more latency, in comparison with the change of TSFs. iv) The maximal number of PhCs allowed in the network is found to be 13, otherwise, system error will occur.

At the end, the latency on varying the number of clusters is discussed. In the Figure 5.15, the effect on multiple clusters or routers in the network is illustrated. Two PhyCs are included per cluster in these cases, and the TSF is ten. The experimental results reveal that scaling up the number of clusters has a very slight effect on the growths of latency.

If latency is the only constraint in the design, increasing the number of clusters is the best option, to implement more neurons on the FPGA. Actually, if too many clusters and routers are placed in the network, the hardware cost will increase significantly. Consequently, the trade-offs between the hardware resource and time constraints should be defined. In the following sections, the resource usage will be discussed, according to

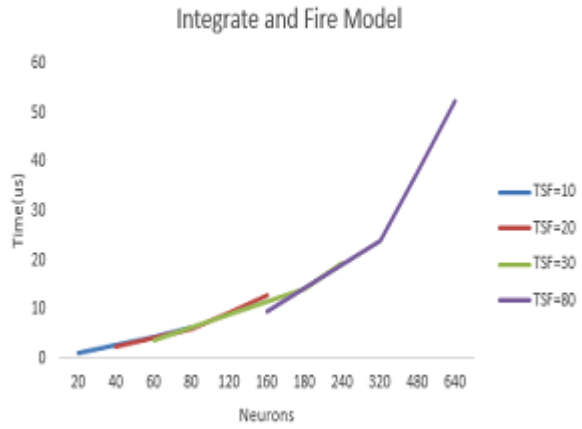


(a) Hodgkin-Huxley model

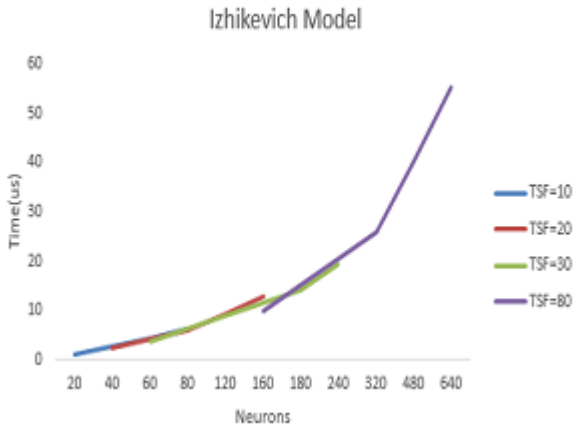


(b) Hodgkin-Huxley model

Figure 5.13: The effect on latency by varying TSFs (left), and varying PhyCs (right), in the neuron network.



(a) Integrate and Fire model



(b) Izhikevich model

Figure 5.14: The effect on latency by varying PhyCs in the neuron network.

multiple logic set-ups. Subsequently, the optimal configurations of neurons models are explored.

5.2.5 Resource Usage

In this thesis, the system is designed to be implemented in the device xc7vx550tffg1927-2, which belongs to the Virtex7 product family. The hardware utilization on the FPGA can be divided into four different resource types, including look-up tables (LTU), flip-flops (FF), digital signal processors (DSP), and block memories (BRAM). By changing the configuration parameters, the effect on these four resource types usages are evaluated in this section.

The effects on resource usages with varying number of the neurons of the Integrate-and-Fire model are show in the Figure 5.16 and 5.17a, and the utilization estimates of Izhikevich model are in the Figure 5.17b and 5.18. Figure 5.19 and 5.20 are related to the resource usage of Hodgkin-Huxley model. After inspecting these figures, it is found

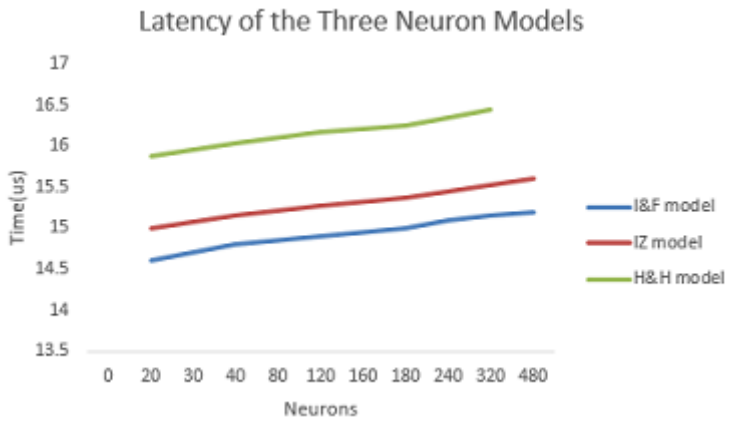
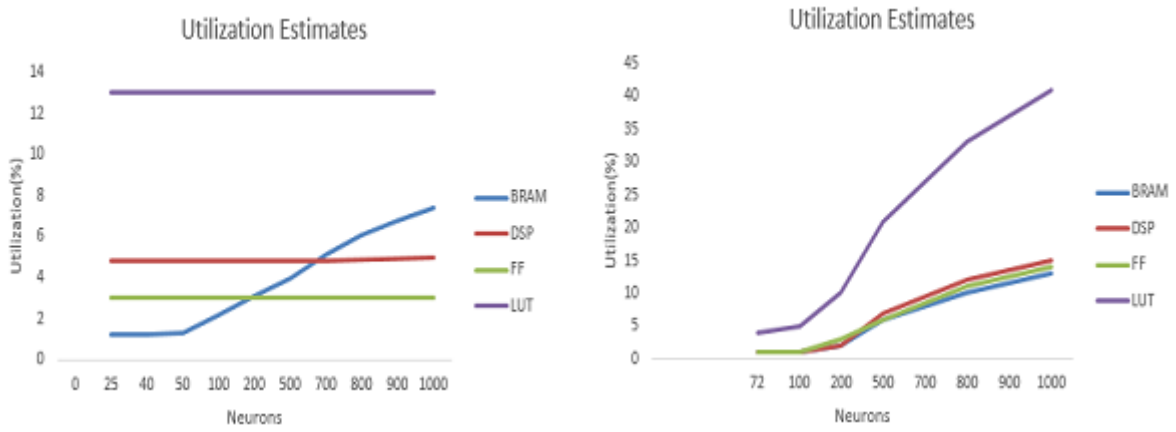


Figure 5.15: The effect on latency by varying the number of clusters in the neuron network.



(a) Integrate and Fire model

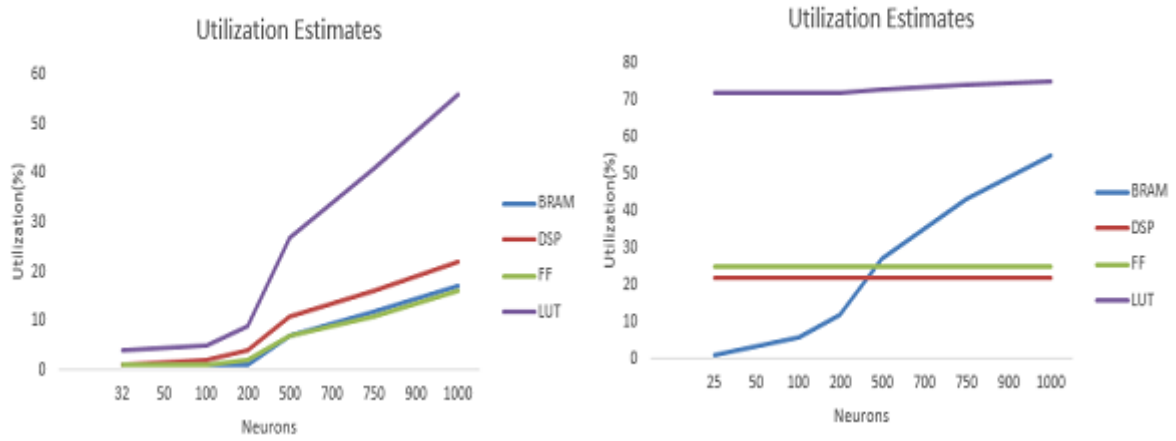
(b) Integrate and Fire model

Figure 5.16: The effect on resource usage by varying TSFs (left), the amount of PhyCs (right) in the Integrate-and-Fire model.

that the corresponding trends on the change of these four resource types, are almost same among three neuron models. In the Figure 5.16a, 5.17b and 5.19a, when adding the value of TSFs in the system, there is nearly no effect on the resource usage, except the BRAM. The amount of BRAM increases due to extra memories needed to store the results of more neurons.

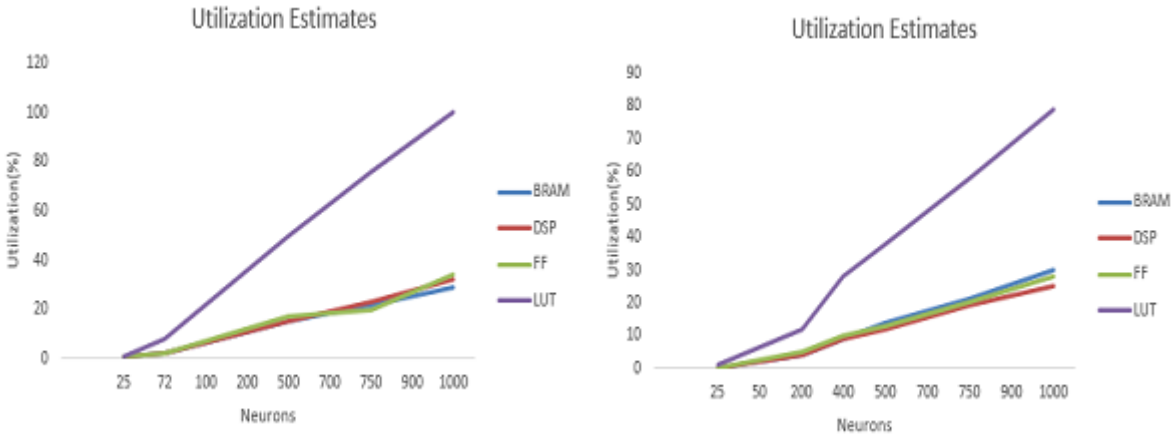
Figure 5.16b, 5.18a, 5.19b, and Figure 5.17a, 5.18b, 5.20 are about varying the amount of Phycs or clusters in the network, respectively. When placing more PhyCs or clusters individually in the network, the hardware utilization of four resource types increase nearly linearly with the growth of neurons (*e.g.,* the original case simulates 500 neurons. After double the amount of clusters, 1,000 neurons are implemented in the system. The hardware utilization of BRAM, DSP, FF, and LUT, increase from 14%, 12%, 13%, 38% to 30%, 25%, 28%, 79%, respectively. The effect is nearly two times.

According to the experimental results, adding the value of TSFs is the most economic way to implement more neurons in the network, while just taking consideration



(a) Integrate and Fire model (b) Izhikevich model

Figure 5.17: The effect on resource usage by varying the number of clusters in the Integrate and Fire model (left), TSFs in the Izhikevich model (right).

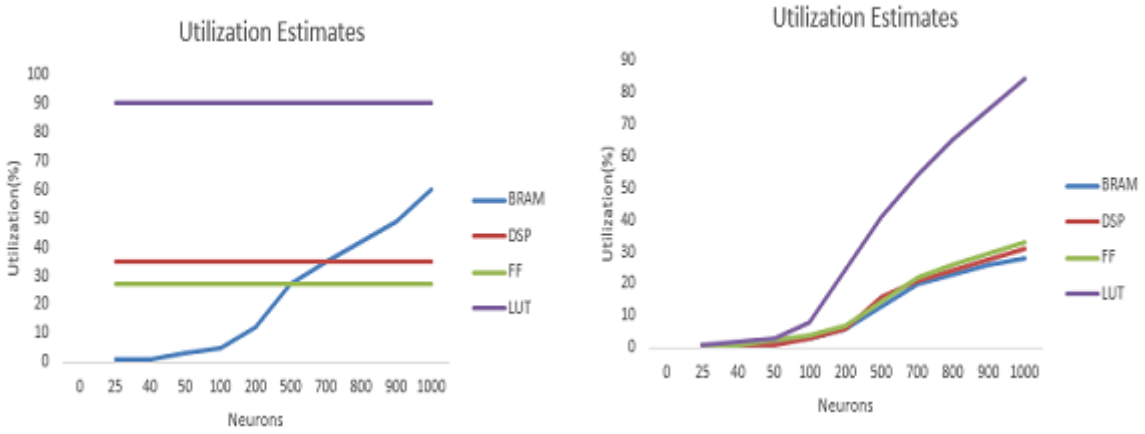


(a) Izhikevich model (b) Izhikevich model

Figure 5.18: The effect on resource usage by varying the amount of PhyCs (left), clusters (right) in the Izhikevich model.

on the limitation of hardware resources. In this situation, only additional BRAM is needed, and the extra requirements of the other three types can be ignored. However, this method is limited by the available amount of BRAM on the FPGA, as well as the time constrain. Consequently, varying the number of PhyCs, or clusters is also adopted to increase the total neurons modeled in the network. Table 5.1 gives the examples of hardware utilization on the PhyCs and clusters, while the Network size is certain. Generally, adjustment of clusters costs more hardware resources than varying PhyCs. This difference is specially on the cost of LUTs. By placing extra clusters in the design, more routers are created, which raises the demand of LUTs.

At the end, the differences on the resource usage among three models are explored, due to the complexity of model design. i.e., Izhikevich model is derived from the



(a) Hodgkin-Huxley model (b) Hodgkin-Huxley model

Figure 5.19: The effect on resource usage by varying the TSFs (left), the amount of PhyCs (right) in the Hodgkin-Huxley model.

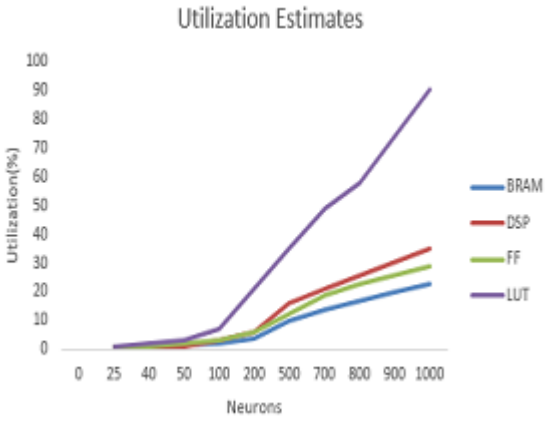


Figure 5.20: The effect on resource usage by varying the number of clusters in the Hodgkin-Huxley model.

Integrate and Fire model, and improved with its own characteristics (e.g. STDP) [29]. As a result, additional registers and memories are required, to store the delay information and STDP parameters. In addition, more look-up tables are created to deal with the increasing communications between the neighbours. Table 5.2 lists the resource

Model	Cluster	PhyC	TSF	BRAM	DSP	FF	LUT
Integrate and Fire model	2	4	25	1%	4%	2%	7%
	4	2	25	2%	4%	3%	10%
Izhikevich model	2	4	25	2%	3%	2%	7%
	4	2	25	4%	5%	5%	14%
Hodgkin-Huxley model	2	4	25	3%	4%	3%	10%
	4	2	25	6%	7%	6%	18%

Table 5.1: The example of effect on varying the number of PhyCs and clusters within the same network size.

utilization of multiple configured designs for the simulated models. Within the same configuration, Hodgkin-Huxley model, the most biophysically-meaningful (complex) neuron model, costs more hardware resources than the other two models. On the contrary, Integrate-and-Fire model requires fewest resources.

Model	Cluster	PhyC	TSF	BRAM	DSP	FF	LUT
Integrate and Fire model	4	10	50	15%	20%	12%	49%
	10	2	25	7%	11%	7%	27%
	4	7	18	6%	13%	8%	33%
Izhikevich model	4	10	50	30%	22%	25%	73%
	10	2	25	14%	12%	13%	38%
	4	7	18	15%	15%	17%	50%
Hodgkin-Huxley model	4	10	50	38%	41%	37 %	124%
	10	2	25	18%	21%	16%	55%
	4	7	18	21%	28%	20%	67%

Table 5.2: The example of resource utilization on the three models within the same configurations.

5.2.6 Model configuration

Through the previous evaluations, all three configuration parameters have the effect on both latency and resource utilization, but each with a different degree. Accuracy, latency, and resources are the critical limitations on the expansion of network size in the system.

By changing the TSFs, more neurons are grouped to compute the responses over the same hardware. Only extra memories are required, and the growth of latency has an nearly linear effect. Varying the number of clusters has an opposite effect. The total latency slightly increases by the additional communication cycles, which are caused by the expansion of network. Compared with the other two dimensions, it has the largest effect on the usage of critical resources. Moreover, the result accuracy is mostly dependent on the number of clusters in the system. While scaling up the network, the amount of routers is added in the tree routing network. Due to the routing complexity, increasing the routers possibly causes the timing violations, and eventually packet loss. The effect of PhyCs has a medium degree on both latency and required hardware. It is used to be a supplement of further enlarging the network size in the system.

The aim of this section is to find the maximal amounts of neurons implemented on the FPGA, for the three neuron models. To find the optimal design, firstly the total implementable physical cells ($Total_{PhyC}$) are decided by the critical resources (5.3). Due to the limitation of result accuracy, the factor, maximal number of clusters (φ), is introduced to the design, which is dependent on the accuracy. After dividing $Total_{PhyC}$ by φ, the amount of physical cells per cluster (PPC) can be determined (5.4).

$$Total_{PhyC} < \frac{Critical, Resource}{Resource, PhyC} \tag{5.3}$$

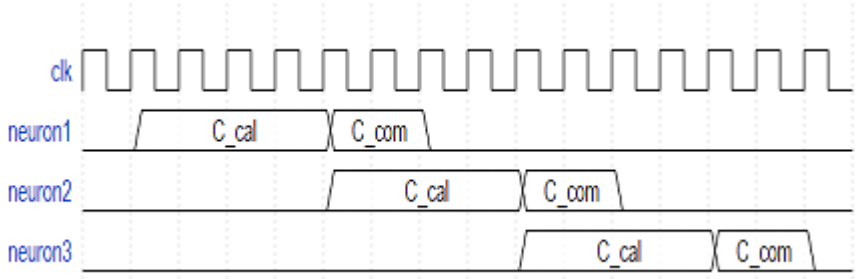


Figure 5.21: An example of timing diagram for PhyC

$$PPC = \lfloor Total_{PhyC} \times \frac{1}{\varphi} \rfloor, \text{ where } \varphi \leq 5 \tag{5.4}$$

To simulate the biological property of neuron models, the latency of physical cells can not be beyond the real brain time ($5ms$). For each neuron, the latency cycles (C_{neuron}) consist of two parts, calculation cycles (C_{cal}) and communication cycles (C_{com}) (5.5). The calculation cycles is the time that cell receiving the response, performing the calculations with parameters, and updating the results in the memories. The duration cost by transmitting the results to corresponding neighbours via the network, is the communication cycles.

$$C_{neuron} = C_{cal} + C_{com} \tag{5.5}$$

Because the system frequency is much higher than the human being [15], multiple cells can be designed as one physical cell and work sequentially inside, as long as the latency of physical cell is within the required time ($5ms$). Figure 5.21 shows that the neurons work in pipeline in the PhyC. Subsequently, the latency of PhyC (C_{PhyC}) can be derived as below (5.6):

$$C_{PhyC} = TSF \times C_{cal} + C_{com} \tag{5.6}$$

To improve the efficiency of system, the C_{PhyC} is aimed to be closed to the biological brain time (C_{brain}) as much as possible, so that more neurons can compute the responses within a given system period (T_{system}) (5.8). From (5.6), an upper bound of the time-share-factor (TSF) can be calculated by taking consideration on the latency of PhyC, calculation cycles, and the communication cycles (5.9).

$$C_{system} = \frac{T_{system}}{CLK_{period}} \tag{5.7}$$

$$C_{PhyC} \leq C_{brain} \leq C_{system}, \text{ where } C_{brain} = 5ms \tag{5.8}$$

$$TSF \leq \lfloor \frac{C_{PhyC} - C_{com}}{C_{cal}} \rfloor \tag{5.9}$$

Then, the total neurons implemented in the system is given by:

$$Total_{neurons} = \varphi \times PCC \times TSF \tag{5.10}$$

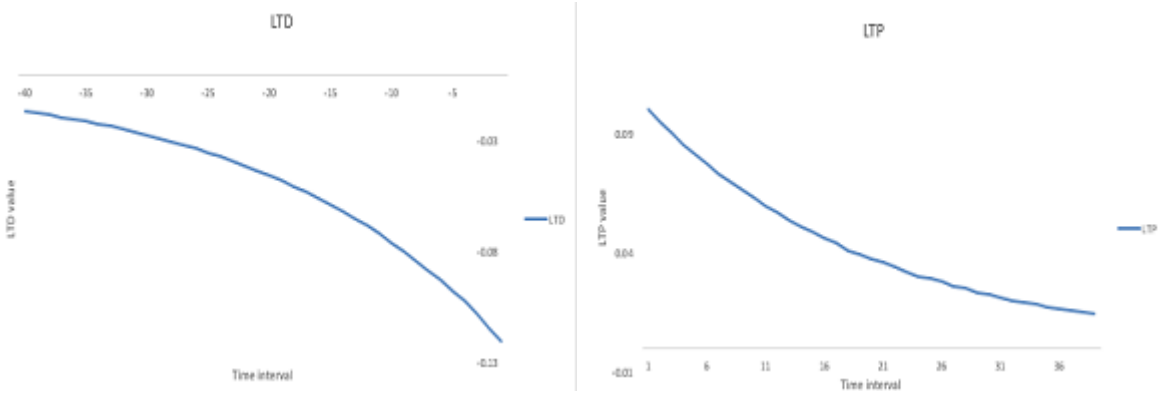
The table 5.3 lists a set of optimal design for the neuron models in the system, where the data of Hodgkin-Huxley model is referenced from the work done by [4]. The Integrate-and-Fire model (Izhikevich model) can implement approximately 5.3 (3.7) times more neurons on FPGA, in comparison with the Hodgkin-Huxley model. It is noted that the maximal number of PhyCs can not be beyond 12 in this design, otherwise, system error will likely issue.

Model	Cluster	PhyC	TSF	BRAM	DSP	FF	LUT	Neurons
Hodgkin-Huxley model	18	2	33	23.60%	35.00%	27.53%	90.01%	1188
Izhikevich model	5	8	110	48%	23%	25%	91%	4400
Integrate and Fire model	5	12	105	42%	31%	20%	94%	6300

Table 5.3: The optimal design of neuron models in the system

5.3 STDP Implementation

The Izhikevich model (2006) includes the characteristic of STDP. Due to the output files can only store the results of potentials, some cout statements are added in the system code, to print out the values of LTD and LTP at each step on the terminal. By recording these values, Figure 5.22a and 5.22b show that the changes of LTD (the post-synapse neuron generates a spike before the pre-synapse neuron) and LTP (the pre-synapse neuron generates a spike before the post-synapse neuron) with varying time interval, respectively. The time interval equals to the time of pre-synapse neuron generating the spike subtracts the time of post-synapse neuron generating the spike. Compared with the reference result [29], it proves that the implementation of STDP can perform correctly in the network.



(a) LTD (b) LTP

Figure 5.22: The effect on LTD (left) and LTP (right) with varying time interval, time interval equals to time of pre-synapse neuron generating a spike - time of post-synapse neuron generating a spike

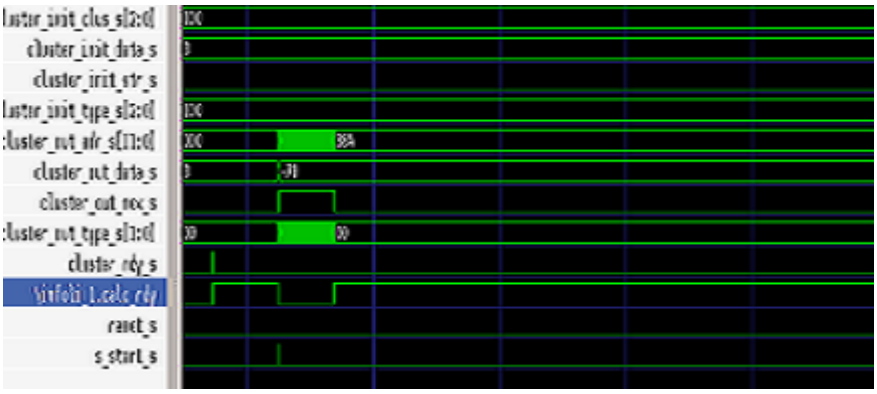


Figure 5.23: A wave diagram of original design

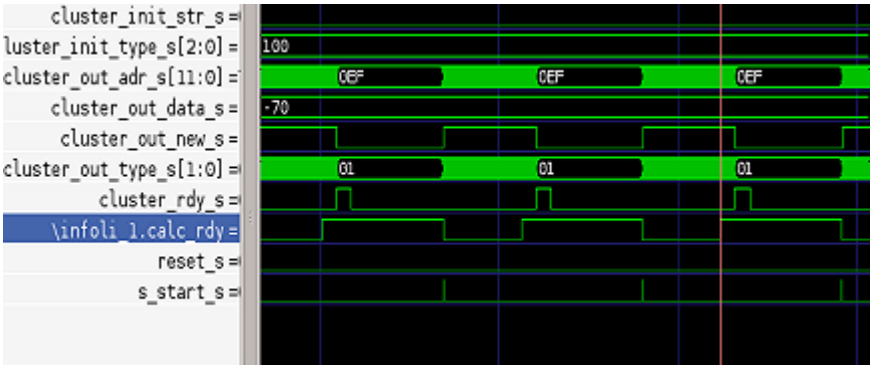


Figure 5.24: A wave diagram after double the buffer depth

5.4 Buffer Depth

According to the analysis of latency and resource usage, increasing the number of clusters is nearly linear with the growth of required critical resources, and has little effect on the latency. Consequently, if the time constant is critical in the design, increasing the number of clusters in the network is an appreciate choice to implement more neurons on FPGA. However, in this design, the maximal amount of clusters is specified to 5. When the logic set-up including more than 5 clusters, the simulation results are likely incorrect, due to the packet loss. The reason is that, while increasing the clusters, more routers are created non-linearly in the tree routing network; Due to the bad routing mechanism (broadcasting), the buffers of routers are likely to overflow and many packets have to be discarded within the given system period. In this section, first it will explore whether expansion of buffer depth can improve the accuracy of system. Next, the relationship between them will be discussed.

The Figure 5.23 gives an example of a test design, where the packets loss issues. In the first iteration, the neurons can normally generate spikes, depending on their own parameters and external stimuli. Then, these new neuron responses are sent to the neighbours via network. At that time, due to too much packets transmitted into network, some packets have to be discarded. Consequently, the destination cells can

not receive the spikes from their neighbours, and the corresponding calculations are suspended. The working mechanism of router is that: when multiple packets arriving concurrently, the router proceeds one of them, and the others are stored in the buffer first. Once the router becomes free, it will fetch the packets from the buffer, and transmit them to the destinations. Based on that, increasing the buffer depth means that the router can handle more packets, and decrease the possibility of packet loss. To prove that, in Figure 5.24, within the same design, the buffer depths are doubled for all the routers in the network. The result reveals that packet loss is successfully solved (Figure 5.24).

In this system, the buffer depth is defined based on the layer where the corresponding router is placed in the tree network. In addition, the size of buffer in upstream port is double times than larger that the downstream ports (5.11)(5.12). If the value of TSF or *PPC* is large, this definition possibly fail to work.

$$Depth_{downstream,ith} = 2^{Base+i}, \text{ where Base } = 1, 2, \cdots N \tag{5.11}$$

$$Depth_{upstream,ith} = 2^{Base+i+1}, \text{ where Base } = 1, 2, \cdots N \tag{5.12}$$

To fix this problem, the worst case (possible maximal number of packets sending to network concurrently) is analyzed first. In the configuration file, the maximal amount of connections to the neighbours, (N), are predefined. Besides the packets with responses, some other packets (Num_{other}) such as external current, are also needed to be forwarded by the routers. By taking considerations on physical cells per cluster(PPC) and time-share-factor (TSF), the maximal amount of packets per cluster can be derived (5.13).

$$Num_{phyc,cluster} = N \times PPC \times TSF \tag{5.13}$$

Subsequently, adding the number of clusters ($Cluster$) in the design, the lower bound of buffer depth can be given.

$$Depth_{downstream,ith} \geq 2^i \times N \times PPC \times TSF + Num_{other} \tag{5.14}$$

$$Depth_{upstream,ith} \geq 2^{i+1} \times N \times PPC \times TSF \tag{5.15}$$

By analyzing these equations, the resources cost of increasing buffer depth is quite large. i.e., in a design (8*2*10, base=3, N=8), the size of buffer need to be scaled up more than 20 times. Figure 5.25 gives an example of resource usage in a logic set-up, with the change of buffer size. While the buffer size scaling up, there are nearly no extra requirements for DSPs or FFs, as well as slight effect on the growth of LUTs. On the other side, the cost of BRAMs keeps growing with the increasing buffer size, due to additional memories are allocated for the buffer.

Besides the cost of resources, latency is another factor which should be taken into consideration. In the designed network, if the buffer size scales up, it is found that the communication latency increases significantly. The reason is that, the packets which should be discarded by the routers, now are pushed into the buffer and wait to be transmitted to their neighbours until the router is free. Figure 5.26 gives an example of

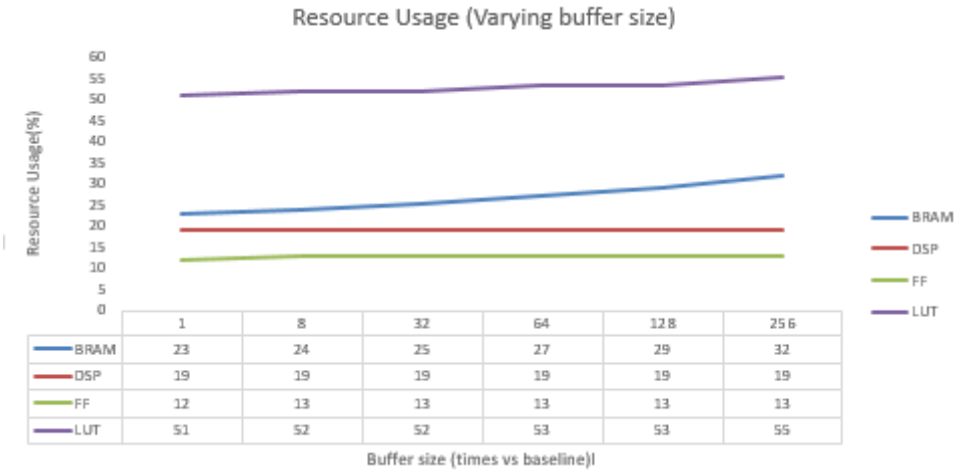


Figure 5.25: An example of resource usage in a test case while varying the buffer size

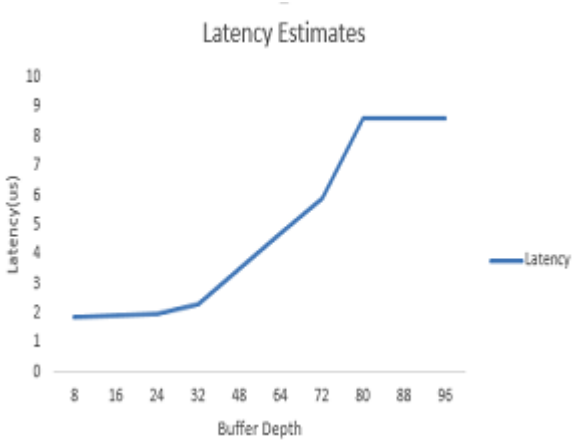


Figure 5.26: An estimate of communication latency while varying the buffer size

design, in which the packet loss is solved after the buffer size is enlarged by ten times. In the Figure 5.26, due to the increasing buffer size, more packets can be handled. Thus, the communication latency also keeps growing. After the depth is large enough, this latency achieves the peak and stop increasing any more.

In the system, the configured set-ups with 5 (or more) cluster controllers cannot be guaranteed to generate the correct results. By increasing the buffer size, this situation can be (partially) improved.

5.5 Summary

In this chapter, the evaluation methods, and some relative tools are introduced first. Then, the neuron properties, simulation time, accuracy, latency, and resource usage are discussed in detail. Depending on the performance analysis, the optimal designs for the simulated neuron models are defined. At the end, the possibility of scaling up the buffer size to improve the accuracy is supplemented.

6 Conclusion and Future Work

In this thesis, I implement several neuron models, representing different trade-offs between the biophysical accuracy and computation complexity. The models with characteristics such as axon conduction delays, spike timing-dependent plasticity (STDP), electrochemical state descriptions, etc, are implemented in a real-time data-flow learning network. These simulated models can be seamless switched in the system for different purposes. e.g. Hodgkin-Huxley model is employed for the research of actual biophysical properties of neuron; Integrate and Fire model can investigate the behavior of network within huge amount of neurons simulated; and, Izhikevich model can be used to inspect the behavior of spike trains (due to tens of parameters and high complexity of computation, Hodgkin-Huxley model is not suitable to generate the specific spike train).

The experimental results indicate that the designed network can well simulate the features of multiple neuron models, since a suitable set of parameters and input are defined. By analyzing the effects on the accuracy, latency, and resource usage, the optimal configurations are defined for the neuron models. The proposed network architecture allow the capacity of over 1,188 (max.6,300, depending on the complexity of neuron model) biophysically accurate neurons in a single FPGA device.

6.1 Future Work

Some recommendations are given to improve the current design below.

Now there are already three neuron models implemented in the system. In future, this network is expected to generally support more neuron models working, e.g. Hindmarsh-Rose model [34], Morris-Lecar model [3], Wilson Polynomial Neuron [10] etc. This challenge has two parts: developing an individual calculation module for the new models, and making a little adjustments on the general architecture to support some special characteristics of new models.

Packet loss is the limit to make use of more free resources, which is caused by the insufficient routing algorithm. In the current network, the routers just simply broadcast the receiving packets to the other nodes. Consequently, the duplication or redundant packets take up too much bandwidth, leading to the overflow of router buffer. To improve that, some smart routing algorithm (e.g. Odd-Even routing algorithm [24, 31]) can be introduced instead. This can help the packets arrive at their destinations in the shortest way, decreasing the workload of routers.

The topology of designed network is the binary tree structure. The advantage of tree network is flexible to expand the size. On the other side, the packets transmitted in this network usually cost more time. Matrix is a potential choice to be instead of current tree structure. The topology of 2D or 3D matrix [36, 40, 12] can better make

use of routing algorithms to optimize the communication between neurons.

The original system [15] can implement multi-FPGA working in a ring topology. However, in order to implement the design on FPGA, this system now is only working on a single FPGA. If the system can work over multiple FPGAs, more neuron cells can be implemented. To improve it, some difficulties are waiting to be fixed. First one is how to define a protocol so that multiple FPGAs can communicate with each other. Then, the control signal should be re-designed for ensuring all FPGAs work synchronously

Bibliography

[1] Abbott, L.F. "Lapique's introduction of the integrate-and-fire model neuron". In: *Brain Research Bulletin* 50 (1999), pp. 303–304.

[2] *Brain Structure*. URL: https://psychbrains.wordpress.com/2014/10/07/brain-structure/.

[3] C. Morris and H. Lecar. "Voltage oscillations in the barnacle giant muscle fiber". In: *Biophys.J* 35 (1981), pp. 193–213.

[4] G.J. Christiaanse. "A Real-Time Hybrid Neuron Network for Highly Parallel Cognitive Systems". In: *MSC Thesis,Delft, The Netherlands: Delft University of Technology* (2016).

[5] O. College. *Anatomy and Physiology*. 2013. URL: Http://cnx.org/content/%20col11496/latest/.

[6] M. van Eijk. "Modeling of Olivocerebellar Neurons using SystemC and High Level Synthesis". In: *IEEE, Biomedical Circuits and Systems Conference* (2014).

[7] Georgios Smaragdos et al. "FPGA-based Biophysically-meaningful Modeling of Olivocerebellar Neurons". In: *Proceedings of the 2014 ACM/SIGDA International Symposium on Fieldprogrammable Gate Arrays. FPGA 14. Monterey, California, USA: ACM* (2014), pp. 89–98. URL: http://doi.acm.org/%2010.1145/2554688.2554790..

[8] H. Gray. "Anatomy of the human body". In: *Lea and Febiger* (1918).

[9] *GTKwave 3.3 Wave Analyzer User's Guide*. URL: http://gtkwave.sourceforge.net/gtkwave.pdf.

[10] H. R. Wilson. "Simplified dynamics of human and mammalian neocortical neurons". In: *J. Theor. Biol* 200 (1999), pp. 375–388.

[11] M. Tyrrell H. Shayani P. Bentley. "A cellular structure for online routing of FPGAs". In: *Evolvable Systems: From Biology to Hardware, Springer Berlin-Heidelbarg* (2008), pp. 273–284.

[12] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano. *Tightly-coupled Multi-layer Topologies for 3-D NoCs*.

[13] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* (1952), pp. 500–544.

[14] " IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual". In: (2006).

[15] J. Hofmann, C. Galuzzi, A. Zjajo, and R. van Leuken. "Multi-chip dataflow architecture for massive scale biophysically accurate neuron simulation". In: *MSC Thesis,Delft, The Netherlands: Delft University of Technology* (2014).

[16] J. R. De Gruijl, P. Bazzigaluppi, M. T. G. de Jeu, and C. I. De Zeeuw. "Climbing fiber burst size and olivary sub-threshold oscillations in a network setting". In: *PLoS Comput Biol* 8.12 (2012).

[17] K. Cheung, S.R. Schultz, W. Luk. "A large-scale spiking neural network accelerator for FPGA systems". In: *International Conference on Artificial Neural Networks and Machine Learning* (2012), pp. 113–120.

[18] Karsten Einwich et al. “Introduction to the SystemC AMS DRAFT standard”. In: *SoCC. IEEE* (2009), p. 446. URL: http://dblp.uni-trier.de/db/conf/%20socc/socc2009.html#EinwichGBV09.

[19] Kit Cheung,Simon R.Schultz, Wayne Luk. “A Large-scale Spiking Neural Network Accelerator for FPGA Systems”. In: *Conference on Artificial Neural Networks and Machine Learning* (2012).

[20] Baptiste Lepilleur. *JSONCpp*. URL: https://github.com/open-source-parsers/jsoncpp.

[21] Levy WB, Steward O. “ Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus”. In: *Neuroscience* 8.4 (1983), pp. 791–797.

[22] M. van Eijk, C. Galuzzi, A. Zjajo, G. Smaragdos, C. Strydis,R. van Leuken. “ESL design of customizable real-time neuron networks”. In: *IEEE International Biomedical Circuits and Systems Conference* (2014), pp. 671–674.

[23] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Network* 10.9 (1997), pp. 1659–1671.

[24] MAK Shrimawale, MA Gaikwad. *Performance Analysis of Odd-Even Routing Algorithm of Network-on-chip Architecture for 2D mesh topology under Bursty Communication Traffic*.

[25] Marcel Beuler,Aubin Tchaptchet, Werner Bonath,Svetlana Postnova, Hans Albert Braun. “Real-Time Simulations of Synchronization in a Conductance-Based Neuronal Network with a Digital FPGA Hardware-Core”. In: *Artificial Neural Networks and Machine Learning* (2012), pp. 97–104.

[26] J.V.Alvarez-Icaza Merolla P.A.Arthur. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345 (2014).

[27] Engene M.Izhikevich. *Which Model to Use for Cortical Spiking Neurons?* URL: http://www.izhikevich.org/publications/whichmod.htm.

[28] Engene M.Izhikevich. “Simple Model of Spiking Neurons”. In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 14.6 (2003).

[29] Engene M.Izhikevich. “Polychronization: Computation with Spikes”. In: *Neural Computation* 18 (2006), pp. 245–282.

[30] Emin Orhan. *The Leaky Integrate-and-Fire Neuron Model*. 2012. URL: http://www.cns.nyu.edu/~eorhan/notes/lif-neuron.pdf.

[31] P Parandkar, JK Dalal,S Katiyal. “Performance Comparison of XY, OE and DY Ad Routing Algorithm by Load Variation Analysis of 2-Dimensional Mesh Topology Based Network-on-Chip”. In: *Bvicams International Journal of Information Technology* (2012).

[32] Peter J. Bentley, Andy M. Tyrrell. “Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA”. In: *Adaptive Hardware and Systems* (2008).

[33] Q.Yu, R.Yan, H.Tang, K.C.Tan, H.Li. “A spiking neural network system for robust sequence recognition”. In: *IEEE Transactions on Neural Networks and Learning Systems* 19.4 (2016), pp. 621–635.

[34] R. M. Rose and J. L. Hindmarsh. "The assembly of ionic currents in a thalamic neuron. I The three-dimensional model". In: *Proc. R. Soc. Lond.B* 237 (1989), pp. 267–288.

[35] Resve Saleh, Michael Jones, Andre Ivanov, Partha Pratim Pande, Cristian Grecu. "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures". In: *IEEE Transactions on Computers* 54 (2005), pp. 1025–1040.

[36] Reza Sabbaghi-Nadooshan, Mehdi Modarressi, Hamid Sarbazi-Azad. "The 2D DBM: An attractive alternative to the simple 2D mesh topology for on-chip networks ". In: *IEEE International Conference on Computer Design* (2008).

[37] Guido van Rossum. *Python Programming Language*. URL: `https://www.python.org/`.

[38] Rumsey CC, Abbott LF. "Equalization of synaptic efficacy by activity- and timing-dependent synaptic plasticity". In: *Neurophysiology* 91.5 (2004), pp. 2273–2280.

[39] H. Adeli S. Ghosh-Dastidar. "Spiking neural networks". In: *International Journal of Neural Systems* 19.4 (2009), pp. 295–308.

[40] Sbastien Le Beux, Ian O'Connor. "Reduction methods for adapting optical network on chip typologies to 3D architectures ". In: *Microprocessors and Microsystems* 37 (2013), pp. 87–98.

[41] *The Integrate-and-Fire Neuron Model*. URL: `http://neuroscience.ucdavis.edu/goldman/Tutorials_files/Integrate%26Fire.pdf`.

[42] W. Gerstner, W.M. Kistler. "Spiking neuron models: single neurons,populations, plasticity". In: *Cambridge University Press* (2002).

[43] J. P. Welsh and R. Llinas. "Some organizing principles for the control of movement based on olivocerebellar physiology". In: *Progress in Brain Research* (1997), pp. 449–461.