HANOI UNIVERSITY OF SCIENECE AND TECHNOLOGY

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

GRADUATION THESIS

Topic:

ASIC IMPLEMENTATION OF OPTIMIZED NEUROSYNAPTIC CORES FOR NEUROMORPHIC ARCHITECTURE

Student: NGUYEN LE TRUNG

Class: ET – E4, 63

Supervisor: ASSOC. PROF. DR. NGUYEN DUC MINH

DR. HOANG PHUONG CHI

Hanoi, 7-2022

HANOI UNIVERSITY OF SCIENECE AND TECHNOLOGY
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

GRADUATION THESIS

Topic:

ASIC IMPLEMENTATION OF OPTIMIZED NEUROSYNAPTIC CORES FOR NEUROMORPHIC ARCHITECTURE

Student: NGUYEN LE TRUNG
Class: ET – E4, 63
Supervisor: ASSOC. PROF. DR. NGUYEN DUC MINH
DR. HOANG PHUONG CHI
Reviewer: .....

Hanoi, 7-2022

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

ĐÁNH GIÁ ĐỒ ÁN TỐT NGHIỆP
(DÀNH CHO CÁN BỘ HƯỚNG DẪN)

Tên đề tài: ASIC Implementation of Optimized Neurosynaptic Cores for Neuromorphic Architectures.
Họ tên SV: Nguyễn Lê Trung MSSV: 20186076
Cán bộ hướng dẫn: PGS. TS. Nguyễn Đức Minh, TS. Hoàng Phương Chi.

STT	Tiêu chí (Điểm tối đa)	Hướng dẫn đánh giá tiêu chí	Điểm tiêu chí
1	**Thái độ làm việc (2,5 điểm)**	Nghiêm túc, tích cực và chủ động trong quá trình làm ĐATN	2,5
		Hoàn thành đầy đủ và đúng tiến độ các nội dung được GVHD giao	
2	**Kỹ năng viết quyển ĐATN (2 điểm)**	Trình bày đúng mẫu quy định, bố cục các chương logic và hợp lý: Bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy, có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn, v.v.	2
		Kỹ năng diễn đạt, phân tích, giải thích, lập luận: Cấu trúc câu rõ ràng, văn phong khoa học, lập luận logic và có cơ sở, thuật ngữ chuyên ngành phù hợp, v.v.	
3	**Nội dung và kết quả đạt được (5 điểm)**	Nêu rõ tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài, các vấn đề và các giả thuyết, phạm vi ứng dụng của đề tài. Thực hiện đầy đủ quy trình nghiên cứu: Đặt vấn đề, mục tiêu đề ra, phương pháp nghiên cứu/ giải quyết vấn đề, kết quả đạt được, đánh giá và kết luận.	5
		Nội dung và kết quả được trình bày một cách logic và hợp lý, được phân tích và đánh giá thỏa đáng. Biện luận phân tích kết quả mô phỏng/ phần mềm/ thực nghiệm, so sánh kết quả đạt được với kết quả trước đó có liên quan.	
		Chỉ rõ phù hợp giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai. Hàm lượng khoa học/ độ phức tạp cao, có tính mới/tính sáng tạo trong nội dung và kết quả đồ án.	
4	**Điểm thành tích (1 điểm)**	Có bài báo KH được đăng hoặc chấp nhận đăng/ đạt giải SV NCKH giải 3 cấp Trường trở lên/ Các giải thưởng khoa học trong nước, quốc tế từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế. **(1 điểm)**	0,5
		Được báo cáo tại hội đồng cấp Trường trong hội nghị SV NCKH nhưng không đạt giải từ giải 3 trở lên/ Đạt giải khuyến khích trong cuộc thi khoa học trong nước, quốc tế/ Kết quả đồ án là sản phẩm ứng dụng có tính hoàn thiện cao, yêu cầu khối lượng thực hiện lớn. **(0,5 điểm)**	
		Điểm tổng các tiêu chí:	10
		Điểm hướng dẫn:	

Cán bộ hướng dẫn
(Ký và ghi rõ họ tên)

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN – ĐIỆN TỬ

ĐÁNH GIÁ ĐỒ ÁN TỐT NGHIỆP
(DÀNH CHO CÁN BỘ PHẢN BIỆN)

Tên đề tài: ASIC Implementation of Optimized Neurosynaptic Cores for Neuromorphic Architectures.
Họ tên SV: Nguyễn Lê Trung MSSV: 20186076
Cán bộ phản biện: ……

STT	Tiêu chí (Điểm tối đa)	Hướng dẫn đánh giá tiêu chí	Điểm tiêu chí
1	**Trình bày quyển ĐATN (4 điểm)**	Đồ án trình bày đúng mẫu quy định, bố cục các chương logic và hợp lý: Bảng biểu, hình ảnh rõ ràng, có tiêu đề, được đánh số thứ tự và được giải thích hay đề cập đến trong đồ án, có căn lề, dấu cách sau dấu chấm, dấu phẩy, có mở đầu chương và kết luận chương, có liệt kê tài liệu tham khảo và có trích dẫn, v.v.	
		Kỹ năng diễn đạt, phân tích, giải thích, lập luận: cấu trúc câu rõ ràng, văn phong khoa học, lập luận logic và có cơ sở, thuật ngữ chuyên ngành phù hợp, v.v.	
2	**Nội dung và kết quả đạt được (5,5 điểm)**	Nêu rõ tính cấp thiết, ý nghĩa khoa học và thực tiễn của đề tài, các vấn đề và các giả thuyết, phạm vi ứng dụng của đề tài. Thực hiện đầy đủ quy trình nghiên cứu: Đặt vấn đề, mục tiêu đề ra, phương pháp nghiên cứu/ giải quyết vấn đề, kết quả đạt được, đánh giá và kết luận.	
		Nội dung và kết quả được trình bày một cách logic và hợp lý, được phân tích và đánh giá thỏa đáng. Biện luận phân tích kết quả mô phỏng/ phần mềm/ thực nghiệm, so sánh kết quả đạt được với kết quả trước đó có liên quan.	
		Chỉ rõ phù hợp giữa kết quả đạt được và mục tiêu ban đầu đề ra đồng thời cung cấp lập luận để đề xuất hướng giải quyết có thể thực hiện trong tương lai. Hàm lượng khoa học/ độ phức tạp cao, có tính mới/ tính sáng tạo trong nội dung và kết quả đồ án.	
3	**Điểm thành tích (1 điểm)**	Có bài báo KH được đăng hoặc chấp nhận đăng/ đạt giải SV NCKH giải 3 cấp Trường trở lên/ Các giải thưởng khoa học trong nước, quốc tế từ giải 3 trở lên/ Có đăng ký bằng phát minh sáng chế. **(1 điểm)**	
		Được báo cáo tại hội đồng cấp Trường trong hội nghị SV NCKH nhưng không đạt giải từ giải 3 trở lên/ Đạt giải khuyến khích trong cuộc thi khoa học trong nước, quốc tế/ Kết quả đồ án là sản phẩm ứng dụng có tính hoàn thiện cao, yêu cầu khối lượng thực hiện lớn. **(0,5 điểm)**	
		Điểm tổng các tiêu chí:	
		Điểm phản biện:	

Cán bộ phản biện
(Ký và ghi rõ họ tên)

PREFACE

Artificial intelligence (AI) applications are now becoming popular and bringing many values to all aspects of life. However, AI computing in current technology is still artificial and less energy efficient than natural systems like our brains. As a result, the AI computation time, especially the training part, is very long and consumes a lot of power. With the end of Moore's Law (the size of the circuit can't get any smaller) and the need for AI computing to grow exponentially, the limitations of current AI technology will become increasingly apparent. Therefore, in the future, AI computers will have to rely on structures closer to nature, or Natural Intelligence (NI).

Nearly all AI systems today are based on computers with the von Neumann architecture (Assoc. John von Neumann – considered the inventor of the computer), all calculations are performed in the central Arithmetic Logic Unit (ALU) (also known as the core), separate from memory. The calculation of current AI models is based on this ALU, and the information in an AI model is calculated on the basis of real numbers. For example, it will take up to 10,000,000,000,000 (ten trillion) objects to be computed with a neural network model with 10 million weights (not too large) and 1 million sample images, consuming a lot of time and energy.

However, a natural brain does not work like that; the calculations take place in each part of the brain itself. Specifically, neurons in the brain will communicate with each other by pulses through axons and their synapses, dendrites. These impulses are of equal magnitudes - that is, now, instead of information being encoded in the direction of magnitudes (real numbers), they are encoded in the time dimension (sparse or dense of the pulses). At this time, the calculation is much simpler, saving more energy.

Spiking neural network is an AI model describing the biological brain. Scientists have also been researching and developing hardware platforms to be able to deploy this model.

Before presenting the content of the project report, I would like to express my sincerest thanks to Assoc. Prof. Dr. Nguyen Duc Minh and Dr. Hoang Phuong Chi, who directly guided and provided me with documents during the project implementation. I would also like to thank my friends in the Neuron Science research group of the EDABK

lab for their great help and the teachers in the EDABK lab for lending me the equipment to complete this project.

Due to time constraints and limited knowledge, the report is not immune to some minor errors. Therefore, I look forward to receiving comments from teachers and friends to improve the topic.

DECLARATION OF AUTHORSHIP

I am Nguyen Le Trung; student ID is 20186076; student of class ET-E4, 2018–2022 school year. My supervisor are Assoc. Prof. Dr. Nguyen Duc Minh and Dr. Hoang Phuong Chi. I hereby declare that all the content presented in the project *ASIC Implementation of Optimized Neurosynaptic Cores for Neuromorphic Architectures* is the result of my team's research (including Pham Huy Hoang and Vu Hoang Long). The data stated in the project is completely honest, reflecting the actual measurement results. All information cited is subject to intellectual property regulations; The references are clearly listed. I take full responsibility for the content written in this project.

Hanoi, 18 July 2022

Author

Nguyen Le Trung

TABLE OF CONTENT

ABBREVIATIONS

TABLE OF FIGURES

LIST OF TABLES

ABSTRACT

The purpose of this project is to follow in the footsteps of previous studies in the field of Spiking neural networks, to find ways to implement a more efficient, more energy-efficient design onto the Application Specific Integrated Circuit (ASIC) platform to have can create a foundation for the design of a true non-von Neumann System on Chip (SoC), where computation takes place at the very components that make up the system.

In this project, I sought to take advantage of existing theories and architecture, redesign, optimize, test, and synthesize them on the ASIC platform to make something new.

I built a new architecture that is more optimized and synthesized it on the ASIC platform with an outstanding performance compared to the existing architecture.

This project report is divided into 6 main chapters as follows:

CHAPTER 1. INTRODUCTION

CHAPTER 2. RANC: Reconfiguration Architecture for Neuromorphic Computing

CHAPTER 3. SNN FULLY PARALLEL ARCHITECTURE

CHAPTER 4. REDESIGN SNN FULLY PARALLEL ARCHITECTURE

CHAPTER 5. SNN TRAINING METHODOLOGY

CHAPTER 6. ASIC DESIGN METHODOLOGY

CHAPTER 1. INTRODUCTION

In the current era, machine learning and artificial intelligence are growing, along with the demand for performance, and the cost of supporting hardware devices is also increasing. For that reason, ideas about hardware architectures that describe biological brains are also emerging. Why biological brains, because in a day, “The brain works with 20 watts. This is enough to cover our entire thinking ability.” [1]. In order to do that, several conceptions need to be adopted, from the most "biological" theories to the most "hardware" theories.

1.1 Computational Neuroscience overview

1.1.1 Definition

According to [2], Computational neuroscience (also known as theoretical neuroscience or mathematical neuroscience) is a branch of neuroscience which employs mathematical models, computer simulations, theoretical analysis and abstractions of the brain to understand the principles that govern the development, structure, physiology and cognitive abilities of the nervous system [3] [4] [5] [6].

Computational neuroscience employs computational simulations to validate and solve mathematical models, and so can be seen as a sub-field of theoretical neuroscience; however, the two fields are often synonymous [3]. The term mathematical neuroscience is also used sometimes, to stress the quantitative nature of the field [7].

Computational neuroscience focuses on the description of biologically plausible Neurons (and neural systems) and their physiology and dynamics, and it is therefore not directly concerned with biologically unrealistic models used in connectionism, control theory, cybernetics, quantitative psychology, machine learning, artificial neural networks, artificial intelligence and computational learning theory [8] [9] [10] [11]; although mutual inspiration exists and sometimes there is no strict limit between fields, [12] [13] [14] [15] with model abstraction in computational neuroscience depending on research scope and the granularity at which biological entities are analyzed.

Models in theoretical neuroscience are aimed at capturing the essential features of the biological system at multiple spatial-temporal scales, from membrane currents, and

chemical coupling via network oscillations, columnar and topographic architecture, nuclei, all the way up to psychological faculties like memory, learning and behavior. These computational models frame hypotheses that can be directly tested by biological or psychological experiments.

1.1.2 Neuromorphic computing

As claimed by [16], Neuromorphic computing [17] [18] [19], is the use of very-large-scale integration (VLSI) systems containing electronic analog circuits to mimic neuro-biological architectures present in the nervous system. A neuromorphic computer/chip is any device that uses physical artificial Neurons (made from silicon) to do computations [20] [21]. In recent times, the term neuromorphic has been used to describe analog, digital, mixed-mode analog/digital VLSI, and software systems that implement models of neural systems (for perception, motor control, or multisensory integration). The implementation of neuromorphic computing on the hardware level can be realized by oxide-based memristors [22], spintronic memories, threshold switches, and transistors [23] [21]. Training software-based neuromorphic systems of spiking neural networks can be achieved using error backpropagation, e.g., using Python based frameworks such as snnTorch [24], or using canonical learning rules from the biological learning literature, e.g., using BindsNet [25].

A key aspect of neuromorphic engineering is understanding how the morphology of individual Neurons, circuits, applications, and overall architectures creates desirable computations, affects how information is represented, influences robustness to damage, incorporates learning and development, adapts to local change (plasticity), and facilitates evolutionary change.

Neuromorphic engineering is an interdisciplinary subject that takes inspiration from biology, physics, mathematics, computer science, and electronic engineering [21] to design artificial neural systems, such as vision systems, head-eye systems, auditory processors, and autonomous robots, whose physical architecture and design principles are based on those of biological nervous systems [26]. It was developed by Carver Mead [27] in the late 1980s.

A neuromorphic computer/chip is any device that uses physical artificial Neurons (made from silicon) to do computations. One of the advantages of using a physical

model computer such as this is that it takes the computational load of the processor (in the sense that the structural and some of the functional elements don't have to be programmed since they are in hardware). In recent times, neuromorphic technology has been used to build supercomputers which are used in international neuroscience collaborations. Examples include the Human Brain Project SpiNNaker supercomputer and the BrainScaleS computer.

1.2 Spiking neural networks overview

1.2.1 Definition

Spiking neural network (SNN) is artificial neural network that more closely mimic natural neural network [28]. In addition to Neuronal and synaptic state, SNNs incorporate the concept of time into their operating model. The idea is that Neurons in the SNN do not transmit information at each propagation cycle (as it happens with typical multi-layer perceptron networks), but rather transmit information only when a Membrane Potential – an intrinsic quality of the Neuron related to its membrane electrical charge – reaches a specific value, called the threshold. When the Membrane Potential reaches the threshold, the Neuron fires, and generates a signal that travels to other Neurons which, in turn, increase or decrease their Membrane Potentials in response to this signal. A Neuron model that fires at the moment of threshold crossing is also called a spiking Neuron model [29]. Figure 1.1 shown how the Membrane Potential in a spiking Neuron work.

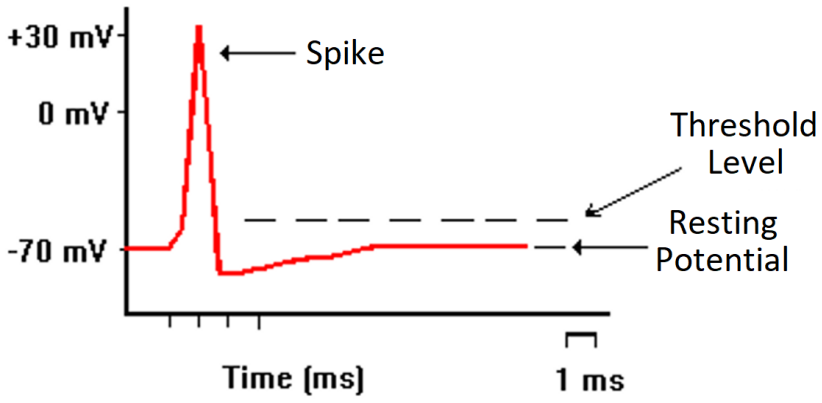


Figure 1.1 Membrane Potential of a spiking Neuron

1.2.2 Model

SNN Neurons are actually built on the mathematical descriptions of biological Neurons. There are two basic groups of methods used to model an SNN Neuron:

- Conductance-based models describe how action Membrane Potentials in Neurons are initiated and propagated:
  - Hodgkin-Huxley model
  - FitzHugh–Nagumo model
  - Morris–Lecar model
  - Hindmarsh–Rose model
  - Izhikevich model
  - Cable theory
- Threshold models generate an impulse at a certain threshold:
  - Perfect Integrate-and-fire
  - Leaky Integrate-and-fire
  - Adaptive Integrate-and-fire

Although all these methods try to describe biological Neurons, the devil is in detail, so SNN Neurons built based on these models might slightly differ.

The most used model for an SNN Neuron is the Leaky Integrate-and-fire (LIF) threshold model as shown in Figure 1.2. This model suggests setting the value in the Neuron to the momentary activation level modeled as a differential equation. Then the Neuron receives incoming spikes that affect the value until it either vanishes or reaches a threshold. If the threshold is reached, the Neuron sends impulses to the downstream Neurons and the value in the Neuron drops below its average [30].

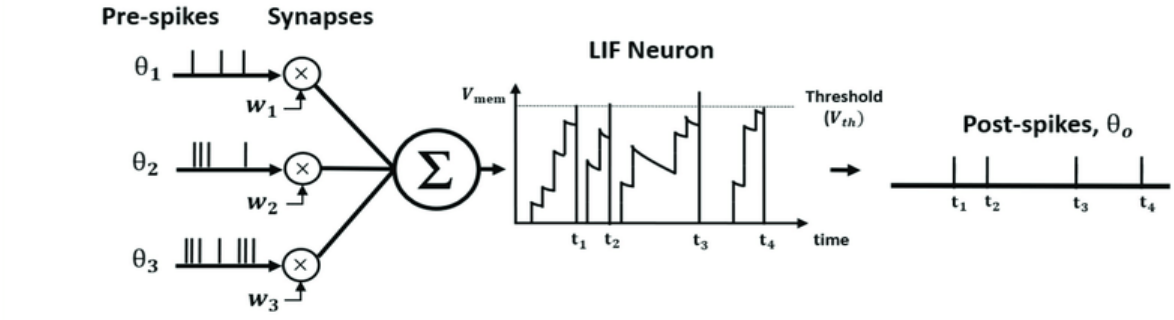


Figure 1.2 Leaky Integrate-and-fire threshold model

Various decoding methods exist for interpreting the outgoing spikes train as a real-value number, relying on either the frequency of spikes (rate-code), the time-to-first-spike after stimulation, or the interval between spikes.

1.2.3 SNN Architectures

Despite an SNN being unique at its concept, it is still a neural network, so SNN architectures can be divided into three groups:

- Feedforward Neural Network is a classical NN architecture that is widely used across all industries. In such an architecture the data is transmitted strictly in one direction – from inputs to outputs, there are no cycles, and processing can take place over many hidden layers. The majority of the modern ANN architectures are feedforward;
- Recurrent Neural Network (RNN) is a bit more advanced architecture. In RNNs connections between Neurons form a directed graph along a temporal sequence. This allows the net to exhibit temporal dynamic behavior. If an SNN is Recurrent, it will be dynamical and have a high computational power;
- In Hybrid Neural Network some Neurons will have feedforward connection whereas others will have recurrent connection. Moreover, connections between these groups might also be either feedforward or recurrent. There are two types of Hybrid Neural Networks that can be used as SNN architecture:
    - Synfire chain is a multilayer net that allows impulse activity to propagate in the form of a synchronous wave of transmission of spike trains from one layer to the other and back;
    - Reservoir computing can be used to build a Reservoir SNN that will have a Recurrent Reservoir and output Neurons.

1.2.4 Training methods

Unfortunately, as of today, no effective supervised interpretable learning method can be used to train an SNN. The critical concept of SNN operations does not allow the use of classical learning methods appropriate for the rest of NNs. Still, scientists are searching for an optimal method. All of that are the reasons why training an SNN might be a challenging task.

1.2.5 Advantages and disadvantages

Spiking Neural Networks have several clear advantages over the traditional NNs:

- SNN is dynamic. Thus, it excels at working with dynamic processes such as speech and dynamic image recognition;
- An SNN can still train when it is already working;
- You need to train only the output Neurons to train an SNN;
- SNN usually has fewer Neurons than the traditional Artificial Neural Network (ANN);
- SNNs can work very fast since the Neurons will send impulses not a continuous value;
- SNNs have increased productivity of information processing and noise immunity since they use the temporal presentation of information.

Unfortunately, SNNs also have two major disadvantages:

- SNNs are hard to train. As of today, there is no learning method designed specifically for this task;
- It is impractical to build a small SNN.

1.2.6 Challenges

In theory, SNNs are more powerful than the current generation of NNs. Still, there are two serious challenges that need to be solved before SNNs will be widely used:

- The first challenge comes from the lack of the learning method developed specifically for the SNN training. Specifics of SNN operations do not allow Data Scientists to effectively use traditional learning methods, for example, gradient descent. Sure, there are Unsupervised biological learning methods that can be used to train an SNN. However, it will be time-consuming and irrelevant since a traditional ANN will learn both faster and better;
- The second one is hardware. Working with SNNs is computationally expensive as it requires solving many differential equations. Thus, you will not be able to effectively work locally without having specialised hardware.

1.2.7 Real Life Applications

SNNs can be applied in various industries, for example:

- **Prosthetics:** As of today, there are already visual and auditory neuroprostheses, which use spike trains to send signals to the visual cortex and return the ability to orient in space to the patients. Also, scientists are working on mechanical motor prostheses that use the same approach. Moreover, spike trains can be supplied to the brain through implanted electrodes and, thereby, eliminate the symptoms of Parkinson's disease, dystonia, chronic pain, and schizophrenia.
- **Robotics:** Brain Corporation based in San Diego develops robots using SNNs, whereas SyNAPSE develops neuromorphic systems and processors.
- **Computer Vision:** Computer Vision is the sphere that can strongly benefit from using SNNs for automatic video analysis. The IBM TrueNorth digital neurochip can help with that as it includes one million programmable Neurons and 256 million programmable synapses to simulate the functioning of Neurons in the visual cortex. This neurochip is often considered the first hardware tool that was specifically designed to work with SNNs.
- **Telecommunications:** Qualcomm is actively researching the possibility of integrating SNNs in telecommunication devices.

1.2.8 Future

There are two opinions on SNNs among Data Scientists: a skeptical and an optimistic one.

Optimists think that SNNs are the future, because:

- They are the logical step in NNs evolution;
- In theory, they are more powerful than traditional ANNs;
- There are already SNNs implementations that show the Membrane Potential of SNNs.

On the other hand, skeptics feel that SNNs are overrated for several reasons:

- There is no learning method designed specifically for SNNs;
- Effective working with SNNs requires specialized hardware;
- They are not commonly used across the industries remaining either a niche solution or a fancy idea;
- SNNs are less interpretable than ANNs;
- There are more theoretical articles on SNNs than practical ones;

- Despite being around for a while, there is still no massive breakthrough in the SNNs sphere.

Thus, the future of SNNs is unclear. SNNs simply need a bit more time and research before they become relevant.

1.3 MNIST dataset overview

The Modified National Institute of Standards and Technology (MNIST [31]) database is a large database of handwritten digits that is commonly used for training various image processing systems [32] [33]. The database is also widely used for training and testing in the field of machine learning [34] [35]. It was created by "re-mixing" the samples from NIST's original datasets [36]. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments [31]. Furthermore, the black and white images from NIST were normalized to fit into a 28×28 pixel bounding box and anti-aliased, which introduced grayscale levels [31].

Figure 1.3 Sample images from MNIST test dataset

The MNIST database contains 60,000 training images and 10,000 testing images [37]. Figure 1.3 shows some sample images from MNIST test dataset. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's

testing dataset [38]. The original creators of the database keep a list of some of the methods tested on it [31]. In their original paper, they use a support-vector machine to get an error rate of 0.8% [39].

The set of images in the MNIST database was created in 1998 as a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively [31].

1.4 ASIC overview

1.4.1 Definition

As explained in [40], Application Specific Integrated Circuits or ASICs are, as the name indicates, non-standard integrated circuits that have been designed for a specific use or application. Generally an ASIC design will be undertaken for a product that will have a large production run, and the ASIC may contain a very large part of the electronics needed on a single integrated circuit. As may be imagined, the cost of an ASIC design is high, and therefore they tend to be reserved for high volume products. Figure 1.4 shown A tray of ASIC chips

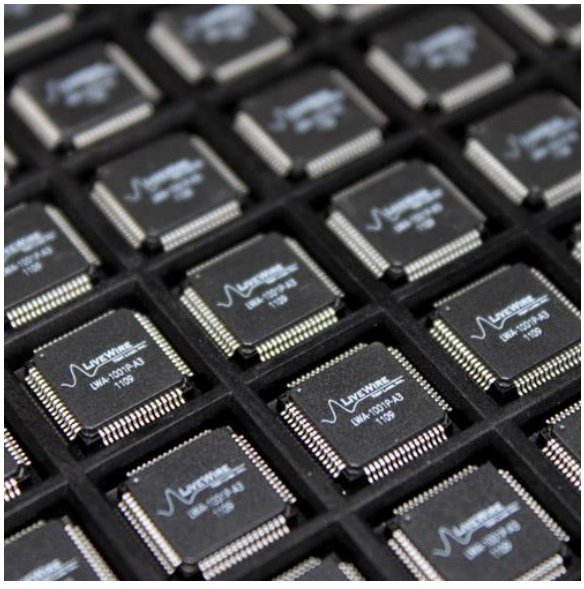


Figure 1.4 A tray of ASIC chips

Despite the cost of an ASIC design, ASICs can be very cost effective for many applications where volumes are high. It is possible to tailor the ASIC design to meet the exact requirement for the product and using an ASIC can mean that much of the overall design can be contained in one integrated circuit and the number of additional

components can be significantly reduced. As a result they are widely used in high volume products like cell phones or other similar applications, often for consumer products where volumes are higher, or for business products that are widely used.

The first ASICs traditionally addressed only logic functions. Now mixed signal ASIC designs can incorporate both analogue (including RF) and logic functions. These mixed signal ASICs are particularly useful in being able to make a complete system on chip, SoC. Here a complete system or product is integrated onto a chip and virtually no other components are required. This makes a mixed signal ASIC design a very attractive proposition for many applications.

ASIC designs offer an attractive solution for many high-volume applications. They enable significant amounts of circuitry to be incorporated onto a single chip. Additional components and board areas would be needed if the circuits were assembled using proprietary chips. Manufacturing costs would be more. With sufficient volume, custom chips in ASICs offer a desirable proposition. In addition to the cost aspects, ASICs may also be used sometimes because they enable circuits to be made that might not be technically viable using other technologies. They might offer speed and performance that would not be possible if discrete components were used. When developing an ASIC, it is often necessary to employ another specialist company to provide the ASIC design service; using their expertise, the design can be undertaken more effectively - in terms of correct functionality, cost and timescale.

1.4.2 Classification

The development and manufacture of an ASIC design including the ASIC layout is a very expensive process. In order to reduce the costs, there are different levels of customisation that can be used. These can enable costs to be reduced for designs where large levels of customisation of the ASIC are not required. Essentially there are three levels of ASIC that can be used:

- **Gate Array:** This type of ASIC is the least customisable. Here the silicon layers are standard but the metallisation layers allowing the interconnections between different areas on the chip are customisable. This type of ASIC is ideal where a large number of standard functions are required which can be connected in a particular manner to meet the given requirement.

- **Standard cell:** For this type of ASIC, the mask is a custom design, but the silicon is made up from library components. This gives a high degree of flexibility, provided that standard functions are able to meet the requirements.
- **Full custom design:** This type of ASIC is the most flexible because it involves the design of the ASIC down to transistor level. The ASIC layout can be tailored to the exact requirements of the circuit. While it gives the highest degree of flexibility, the costs are very much higher and it takes much longer to develop. The risks are also higher as the whole design is untested and not built up from library elements that have been used before.

1.5 Conclusion

CHAPTER 1 has provided full knowledge and basic concepts about Computational Neuroscience, SNN, MNIST dataset, and ASIC; and their applications in the simulation of biological neural networks in the form of electronic hardware. In addition, the advantages/disadvantages and prospects of SNN have also been briefly presented to provide a more comprehensive view of the project's future. CHAPTER 2 will continue to present an open-source architecture that simulates the behavior of TrueNorth [41] - a pioneering architecture in the field of SNN simulation on electronic hardware.

CHAPTER 2. RANC: Reconfigurable Architecture for neuromorphic Computing

The new era of cognitive computing brings forth the grand challenge of developing systems capable of processing massive amounts of noisy multi-sensory data. This type of intelligent computing poses a set of constraints, including real-time operation, low power consumption and scalability, which require a radical departure from conventional system design. Brain-inspired architectures offer tremendous promise in this area [41].

Neuromorphic architectures have been introduced as platforms for energy efficient spiking neural network execution. The massive parallelism offered by these architectures has also triggered interest from nonmachine learning application domains. In order to lift the barriers to entry for hardware designers and application developers RANC: a Reconfigurable Architecture for Neuromorphic Computing is presented, an open-source highly flexible ecosystem that enables rapid experimentation with neuromorphic architectures in both software via C++ simulation and hardware via Field-programmable gate array (FPGA) emulation [42].

IBM's TrueNorth targets a neuromorphic on-chip design and uses a "globally synchronous-locally asynchronous" methodology to construct the network [41]. Replacing the methodology with the fully synchronous one and with some modifications such as inserting Router's buffers, disabling the stochastic mode of Neuron Block, and reducing the number of states in Token Controller's state machine, RANC successfully deploy their new network on FPGA and achieve a notable feat [42]. In this part, a detailed description of RANC hardware architectures will be depicted.

2.1 High level description

As neuromorphic architectures are designed to mimic the behavior of biological neurons in the brain, the primary data unit is the “spike”, which is typically modeled as a Dirac delta function occurring as a part of a time series. In digital neuromorphic architectures, spikes are usually encoded as either a digital 1 or 0. These spikes are sent to neuron units that then react accordingly by either producing more spikes or remaining dormant based on configuration parameters. Individual neuron units communicate by

sending each other spikes, and in this way, computation may be performed over time by sending carefully crafted sequences of spikes into a richly connected network of neuron units and observing the resulting behavior.

2.1.1 Hardware – Biological model

SNN network tries to simulate as accurately as possible the structure and functioning of the human brain, including almost all the basic components of the brain, represented as digital hardware blocks.

The brain's ability to store data or memory is formed by connecting Neurons and Axons and setting weights based on the set of training parameters, rather than the set of training parameters. there are memory blocks like Random Access Memory (RAM), and Read-only memory (ROM) common to Von Neumann architecture.

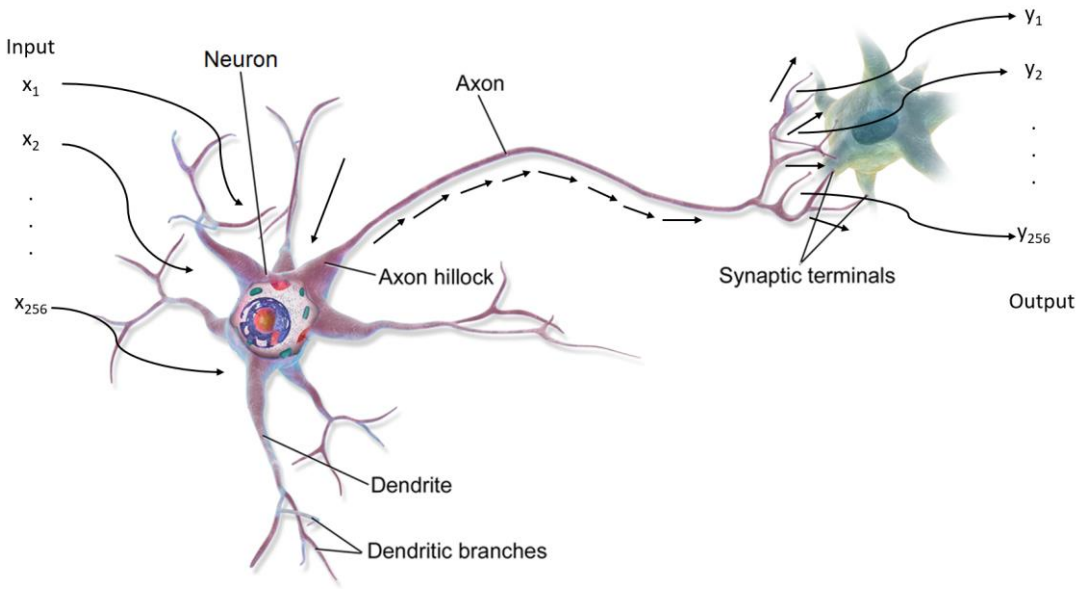


Figure 2.1 Spiking Neuron model

As depicted in Figure 2.1, a Neuron can receive multiple inputs from its Dendrites (256 inputs in Truenorth architecture) and fire a single output through its Axon. This Axon can connect to many other Neurons through their Dendrites (up to 256 Neurons in Truenorth architecture), these connections are called *synapses*.

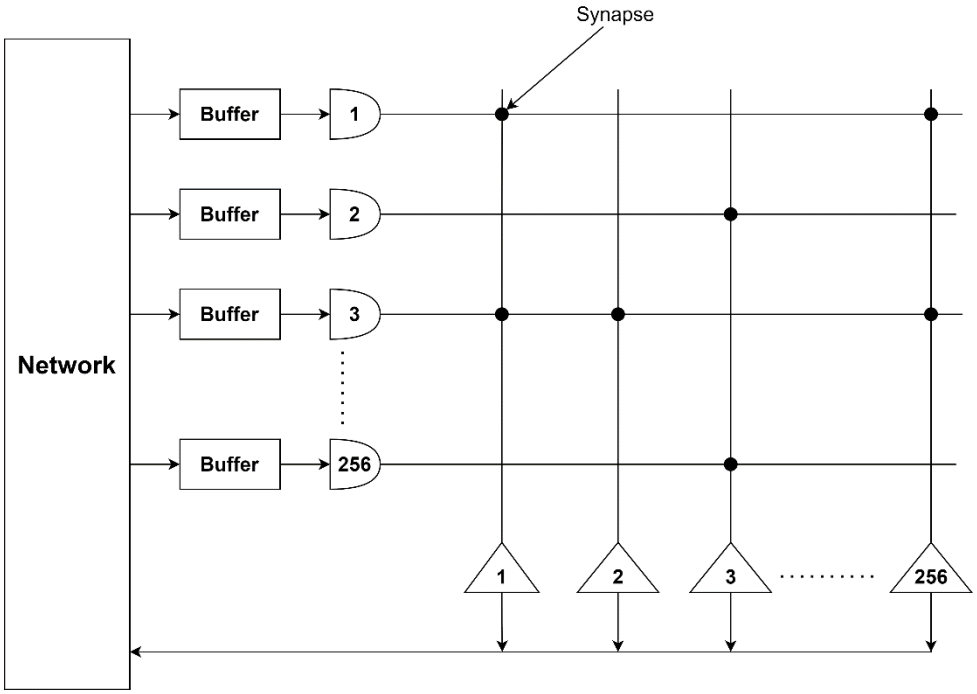


Figure 2.2 Biological neural network correlation model

Figure 2.2 describes the biological neural network correlation model. Specifically, this figure depicts a cluster of 256 Neurons that each Neuron can connect to up to 256 Axons, these Neurons receive spikes as inputs from the network, process, and shoot spike (if any) to other clusters according to the principle of Leaky Integrate-and-fire model. This whole process closely describes the theory mentioned above.

The basic components of a biological neural network model:

- **Neuron**: integrates Membrane Potential, when this value exceeds a certain threshold, the Neuron will release a spike.
- **Dendrite**: lines branching from a Neuron.
- **Axon**: the nerve pathways that connect the Neurons to each other.
- **Synapse**: connection point between an Axon and a Denrite.

Summary:

- A Neuron can receive stimuli from multiple Axons through its Denrites.
- One Axons can connect to many Neurons to propagate the stimulus.

From that, we can obtain some general equations for the whole system:

Integrated Membrane Potential equation:

$$\mu = v_i(t-1) + \left(\sum_j spike_j \times w_j^{\tau_j} \right) + \ell_i,\ where\ j \in \Omega \qquad (2.1)$$

Membrane Potential equation:

$$v_i(t) = \begin{cases} r_i^+ & if\ \mu \geq v_i^+ \\ r_i^- & if\ \mu < v_i^- \\ \mu & otherwise \end{cases} \qquad (2.2)$$

Spike equation:

$$s_i(t) = \begin{cases} 1 & if\ \ v_i \geq v_i^+ \\ 0 & otherwise \end{cases} \qquad (2.3)$$

Where,

Ω	is the set of all Axons that connect to the current Neuron
$spike_j$	indicates whether or not there is an incoming spike at j^{th} Axon
μ	is the integrated Membrane Potential of the current Neuron
$vi\ (t)$	is the Membrane Potential of Neuron i at time t
τ_j	is the Neuron instruction corresponding to j^{th} Axon
$w_j^{\tau_j}$	is the τ_j^{th} kind of weight of Axon j
ℓ_i	is the leak potential of i^{th} Neuron
s_i	indicates whether or not there is an outgoing spike at *i-th* Neuron
v_i^+	is the positive threshold
v_i^-	is the negative threshold
r_i^+	is the positive reset potential
r_i^-	is the negative reset potential

2.1.2 RANC Architectural description

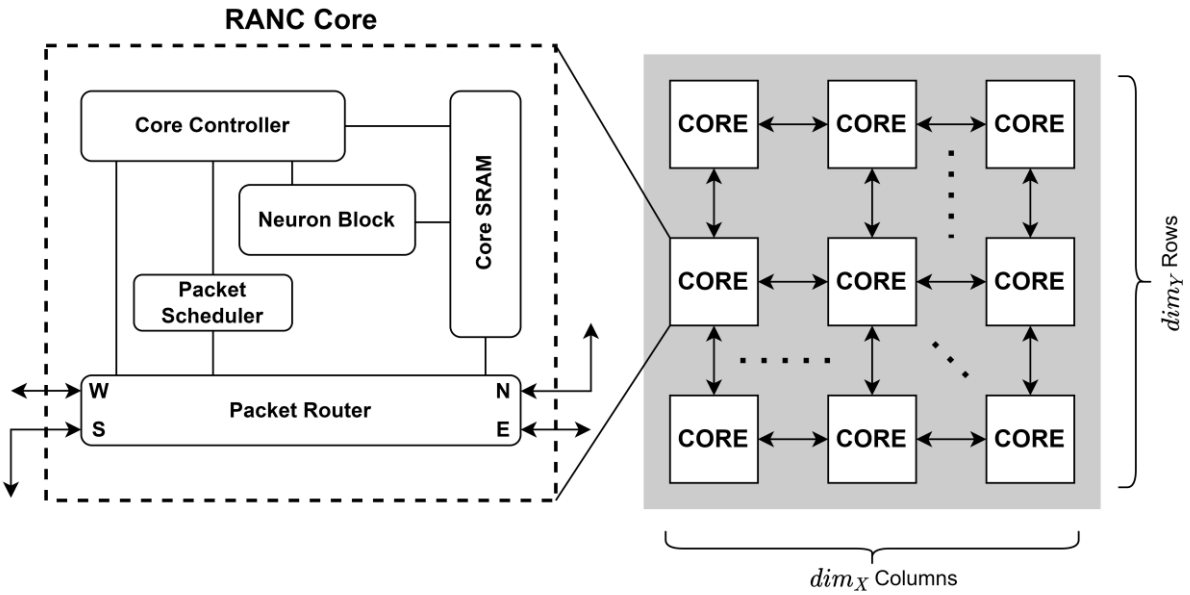


Figure 2.3 High level architectural overview of RANC components.

The overall platform shown in Figure 2.3 is a 2D mesh-based network-on-chip (NoC) composed of cores with five key components: Neuron Block, Core Controller, Core SRAM, Packet Router, and Packet Scheduler. This set of five components is derived by generalizing and parameterizing components of IBM's TrueNorth [41] architecture in an effort to ensure RANC provides a well-tested set of baseline neuromorphic functionalities. The fundamental neuron computational behaviors are implemented via the Neuron Block, and each Neuron Block is coupled to a Core Controller and Core SRAM that coordinates data transfers and stores configuration parameters respectively. Output spikes from each neuron are routed between Neuron Blocks via the Packet Router, and incoming spikes are scheduled for computation via the Packet Scheduler. Design-level synchronization is accomplished via a global synchronization signal or "*tick*" that ensures all cores stay in lockstep throughout their computations. In this way, these five core components are able to work together to form a foundational basis for neuromorphic computing architectural research while leaving flexibility to alter the fine-grained behavior of each unit independently from the rest of the design. In the following subsections, we will discuss each component in detail. All design parameters associated with RANC are defined in Table 2.1.

Table 2.1 Parameters and variables used throughout the RANC

Type	Symbol	Description
Neuron Param	$v_j(t)$	voltage potential for neuron j at tick t
	v_j^+	positive threshold for neuron j
	v_j^-	negative threshold for neuron j
	w_j^k	weight k for neuron j
	ℓ_j	leak value for neuron j
	r_j^+	positive reset value for neuron j
	r_j^-	negative reset value for neuron j
	$s_i(t)$	spike value on axon i at time t
	τ_i	axon type for axon i
	a_i	the i^{th} axon in a core
	n_j	the j^{th} neuron in a core
NoC Param	f_{tick}	frequency of the global NoC tick
	f_{core}	frequency of a single RANC core
	dim_x	X dimension of NoC grid
	dim_y	Y dimension of NoC grid
	dim_x^{max}	Max range of NoC packet in X direction
	dim_y^{max}	Max range of NoC packet in Y direction
Numeric Param	$N(a)$	number of axons per core (= 256)
	$N(n)$	number of neurons per core (= 256)
	$N(t)$	number of tick slots per core (= 16)
	$N(w)$	number of weights supported per neuron (= 4)
Bitwidth Param	$B(a)$	bits per axon index (i.e. $\log_2(N(a))$)
	$B(n)$	bits per neuron index (i.e. $\log_2(N(n))$)
	$B(t)$	bits to index a single tick (i.e. $\log_2(N(t))$
	$B(w)$	bits to represent a weight (= 9)
	$B(v)$	bits to represent a potential value (= 9)
	$B(\ell)$	bits to represent a leak value (= 9)

2.2 RANC network grid

2.2.1 TOP module

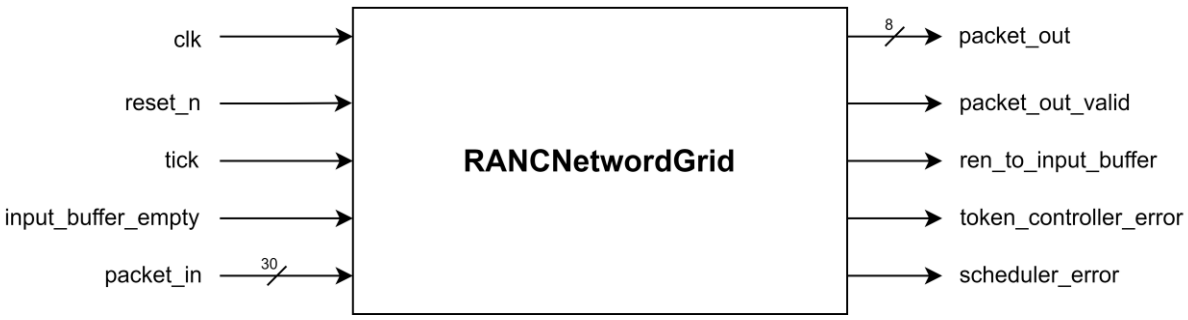


Figure 2.4 TOP module of RANC network grid

Figure 2.4 shows the TOP module of RANC network grid.

2.2.2 Interface signal

Table 2.2 RANC network grid interface signals

Signal name	Width	I/O	Description
clk	1	Input	DTI Clock Signal
reset_n	1	Input	DTI Asynchronous Reset, active LOW
tick	1	Input	Start working, active HIGH
input_buffer_empty	1	Input	Indicate the moment that the input buffer is full
packet_in	30	Input	Input packet to RANC network
packet_out	8	Output	Output packet
packet_out_valid	1	Output	Indicate the moment that there is a new packet
ren_to_input_buffer	1	Output	Request axon_spikes from the storage
token_controller_error	1	Output	Indicate there is error in Core Controller
scheduler_error	1	Output	Indicate there is error in Packet Scheduler

2.2.3 Functional description

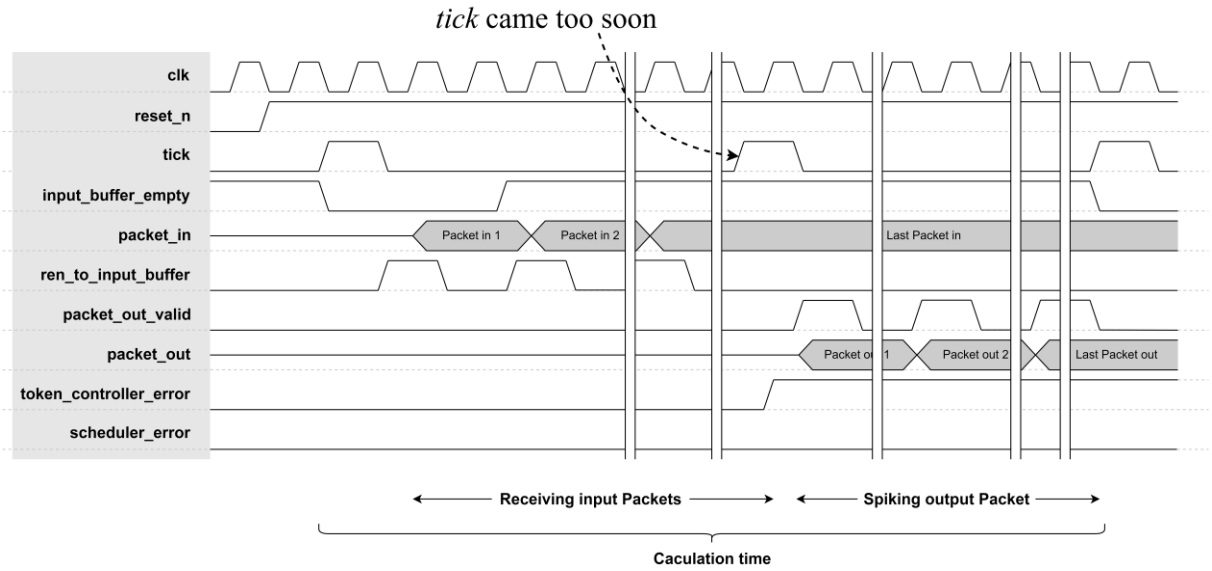


Figure 2.5 RANC network grid waveform diagram

As described in section 2.1.2, a RANC network grid consists of many Cores, and each Core consists of 5 components: Neuron Block, Core Controller, Core SRAM, Packet Router, and Packet Scheduler. When the RANC network grid receives a *tick*, this *tick* will be propagated to the Core Controllers of all Cores in the grid. These Core Controllers will conduct state transition according to the rules that will be mentioned in 2.3.3. The whole calculation process will be automatically controlled, for each certain state, Core Controller will send and receive signals to other components in the Core to which it belongs.

Figure 2.5 shows how the interface signals (as shown in Table 2.1) in RANC network grid behave. Packets sent to the RANC network grid will be stored in a buffer, this buffer sends the *input_buffer_empty* signal to the RANC network grid to signal that it is ready to send the packet, from which the RANC network grid will send the *ren_to_input_buffer* signal back to the buffer. This to signal whether it is ready to receive the packet or not, specifically as follows:

- If *input_buffer_empty* $= 0$, which means that the buffer containing input packets is ready to send packets, the RANC network grid will now send *ren_to_input_buffer* at continuous clock cycles if it is ready to receive packets
- If *input_buffer_empty* $= 1$, it means that the buffer containing input packets is not ready to send packets, at this time, the RANC network grid will not send *ren_to_input_buffer*, no packets will be brought into the network.

For faster computation, the tick frequency (f_{tick}) can be increased. However, if the f_{tick} is too high which can lead to some errors in the calculation, *token_controller_error* and *scheduler_error* have been designed to make error detection easier to properly adjust the ftick:

- While the calculation is in progress, if there is a *tick* coming (f_{tick} is too high), it will rise *token_controller_error* flag
- In case of packets that have not yet reached their destination and a *tick* has occurred, the *scheduler_error* flag will be raised

2.3 RANC Core's components

2.3.1 Neuron Block

As the primary computational component, the Neuron Block emulates a crossbar with 256 input "axons" connected to 256 output "neurons", where 256 and 256 are parameters specified by the user. With this, a single Neuron Block can emulate a total of 256×256 synaptic connections. Additionally, each input axon is associated with a hardcoded index that specifies the location of its associated weight value within the Core SRAM. To emulate each of these Neurons, each Neuron Block contains a basic datapath for mimicking the voltage characteristics of an LIF Neuron model. At a high level, this datapath shown in Figure 2.6 works by having each Neuron maintain a signed running sum known as its Neuron potential (NP). In each cycle, this Neuron potential is either incremented, decremented, or maintained based on a number of potential events. For each input spike that a Neuron receives on one of its axons, the signed weight associated with that axon is accumulated with the current Neuron potential. If, at the end of this process, a Neuron is above its user defined maximum threshold, then that Neuron outputs a spike to its connections and the potential either resets to a static value or subtracts a userdefined value. If, alternatively, a Neuron is below its user defined minimum threshold, that Neuron resets similarly either back to a static value or it adds a user-defined value to the potential without emitting a spike. Finally, if the Neuron value does not cross either the minimum or maximum threshold, it simply "leaks" or decays by a user-specified value. At the end of its computation, a Neuron's final potential value is written into the core SRAM.

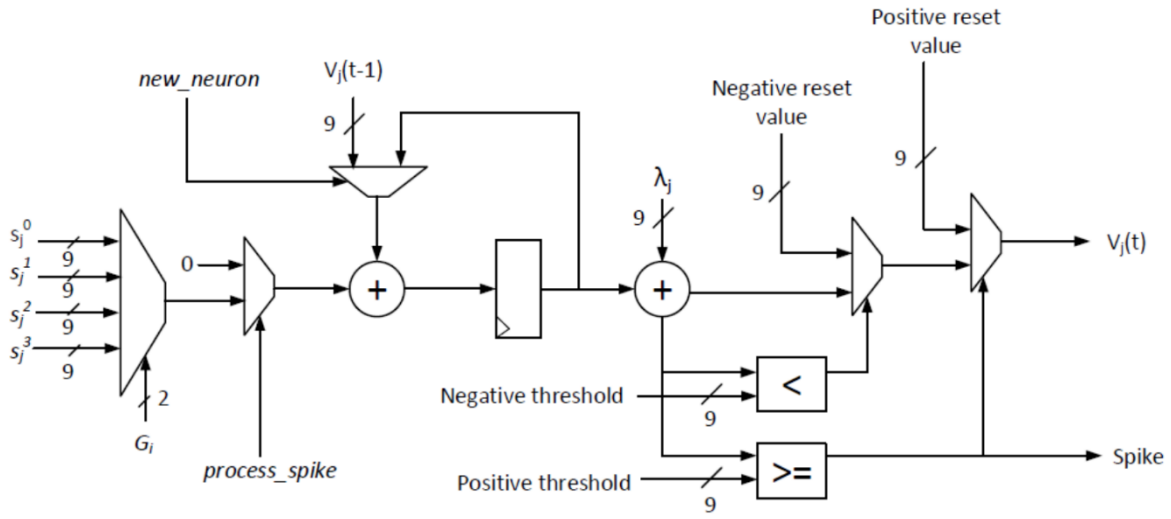


Figure 2.6 RANC Neuron Block

2.3.2 Core Controller

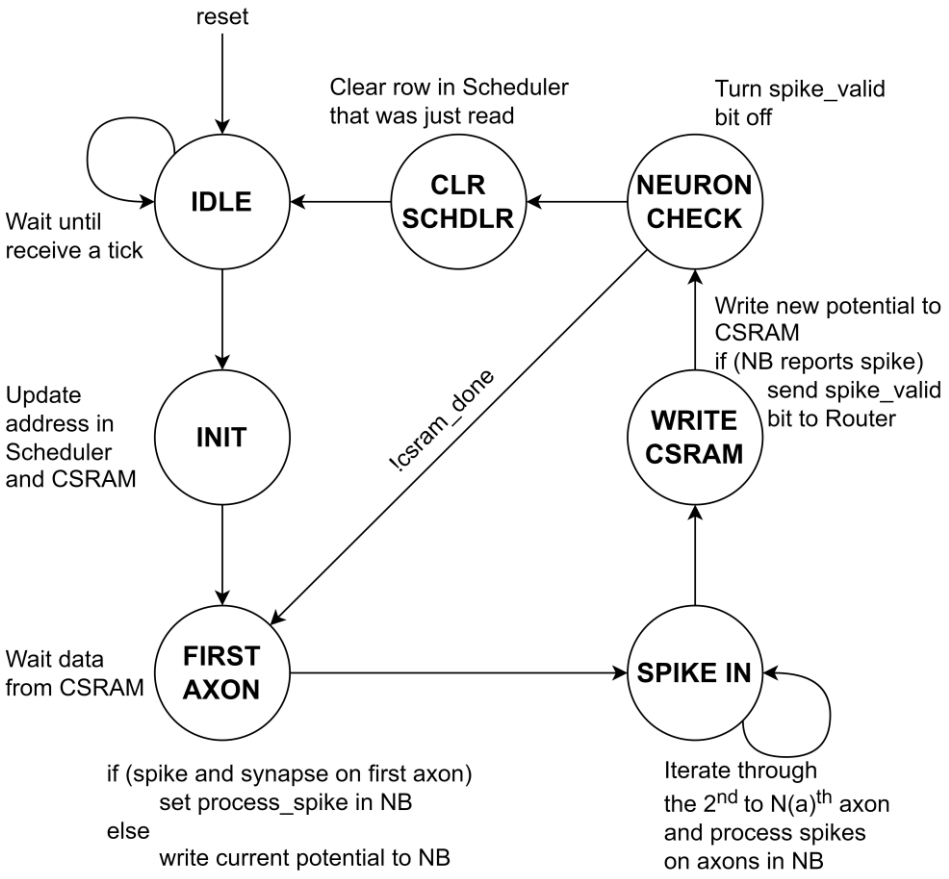


Figure 2.7 RANC Core Controller

The Core Controller, as shown in Figure 2.7, works with the Neuron Block datapath to coordinate memory accesses and data transfers to ensure that all 256 Neurons inside the Core are being emulated faithfully. The waveform diagram of Core Controller was

shown in Figure 2.9. For each Neuron in the Neuron Block, the Core Controller iterates over its associated input Axons (*row_count*). Information regarding the spikes sent to this Neuron are retrieved from the Core SRAM, and for each Axon that delivers a spike, the Core Controller checks if the Axon-Neuron crossbar contains a connection at this location (*synapse* signal). If a connection is present, the weight associated with that Axon is sent to the Neuron Block datapath for accumulation into the Neuron potential. Once all input spikes are processed for a given Neuron, the Controller checks if this Neuron has produced an output spike. If it has, the Controller sends a *spike_out_valid* signal to the Router for it to enqueue this spike (*packet_out*) for delivery to its destination. Once all 256×256 synaptic connections are processed by the Core Controller, it proceeds to an IDLE state until the next synchronization tick occurs. If the synchronization signal occurs before computation completes (i.e. f_{tick} is too fast), an *error* flag is raised to notify the user that output may be corrupted.

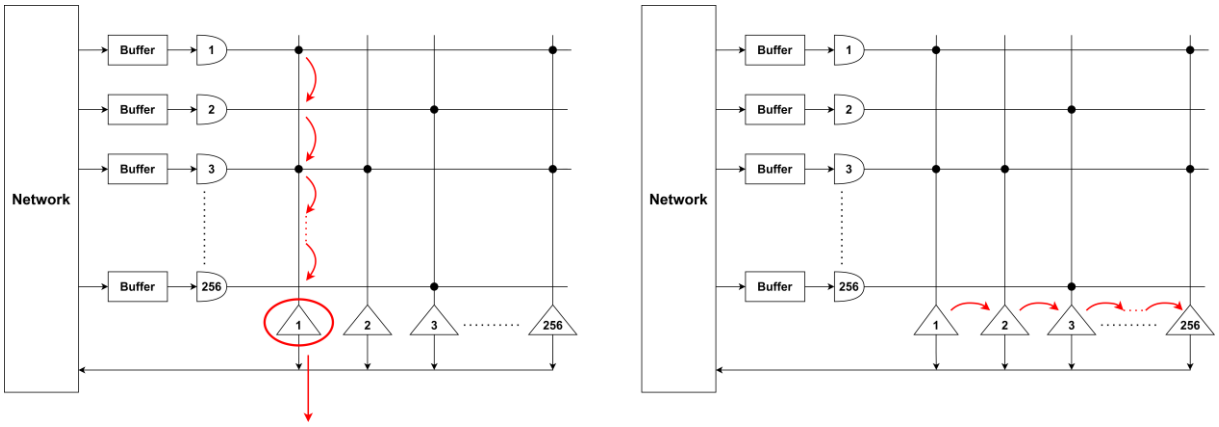


Figure 2.8 RANC Core operating principle: (a) Iterate over each (b) Iterate over each Neuron (right)

Figure 2.8 shows how a Core in RANC architecture behaves when controlled by the Core Controller.

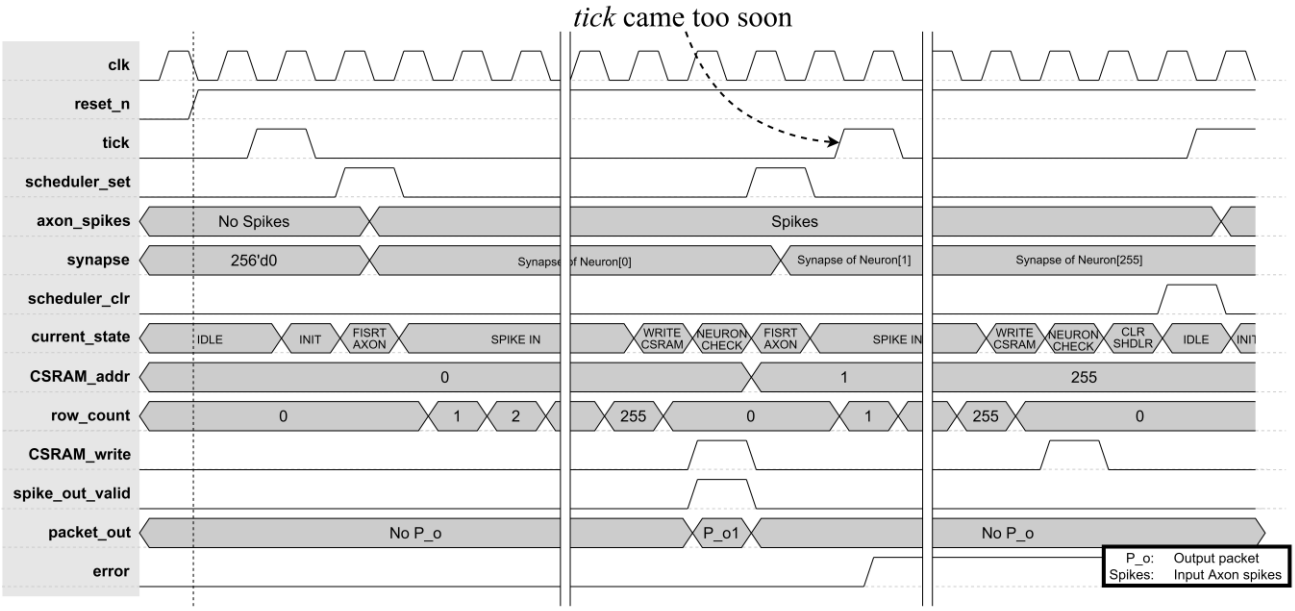


Figure 2.9 RANC Core Controller waveform diagram

Thus, it is possible to estimate the computation time of a RANC Core:

- After receiving a *tick*, Core Controller takes 1 clock cycle to switch from IDLE to INIT ⇒ 1 cycles required
- Then Core Controller performs sequential computation for 256 Neurons by going through FIRST AXON to initial Axon index; in SPIKE IN, for each Neuron, it takes 255 cycles to compute (through 256 Axons), after completing computation for each Neuron, Core Controller goes to WRITE CSRAM to update Neuron potential to CSRAM and NEURON CHECK to check if calculation is done for all 256 Neurons yet => $(255 + 3) \times 256 = 66048$ cycles required.
- After calculating for all Neuron, Core Controller goes to CLR SCHDLR to clear the old data ⇒ 1 clock cycles required

⇒ Thus a conventional architecture based on the RANC design takes a total of:

$$\tau_{computation} = [1 + 66048 + 1] = 66050\ (clock\ cycles) \qquad (2.4)$$

cycles to complete the computation and ready to receive the next *tick*.

2.3.3 Core SRAM

All user-supplied configuration parameters that are relevant for the configuration of each Neuron in a single Core are stored in the Core SRAM. It is defined by a matrix of 256 rows by 368 columns, where 368 is the number of bits required to encode all parameters for a single Neuron (which varies based on user parameter choices, i.e as shown in 2.3), and there are 256 data words for all 256 Neurons. All parameters for each Neuron (weights, connections, current potential, reset values, thresholds, leak value, destination of generated spikes) are stored as a single word in the Core SRAM. As such, each Neuron requires only one read per tick from the Core SRAM. Each of these parameters are assigned to their respective inputs in Figure 2.6, and the Neuron processes all 256 input Axons through states FIRST AXON to NEURON CHECK of Figure 2.7. After computation is complete, the final $v_j(t)$ potential value is updated and the modified word of the Core SRAM is committed back to memory in a single write, after which the Core Controller proceeds on to the next neuron or to an idle state waiting for the next *tick*.

Table 2.3 Core SRAM Parameter Breakdown

Parameter Name	**Bit Width**
Synaptic Connections	256 bits
Potential and Neuron Parameters	82 bits
Spike Destination	26 bits
Delivery Tick	4 bits
Potential & Neuron Parameters	**Bit Width**
Current Potential	9 bits
Reset Potential	9 bits
Weights 0 1 2 and 3	9 bits each
Leak Value	9 bits
Positive/Negative Threshold	9 bits each
Reset Mode	1 bit

2.3.4 Packet Router

Algorithm 2.1 Dimension-order/"XY" packet routing

```
1:  function route (packet, core, dx, dy)
2:      if dx < 0 then
3:          route(packet, core.east, dx + 1, dy)
4:      else if dx > 0 then
5:          route(packet, core.west, dx − 1, dy)
6:      else if dy < 0 then                          // dx == 0
7:          route(packet, core.north, 0, dy + 1)
8:      else if dy > 0 then
```

```
 9:            route(packet, core.south, 0, dy − 1)
10:        else                                          // dx == 0 && dy == 0
11:            core.accept(packet)
12:        end if
13:    end function
```

With the operation of a single Core of Neurons defined, the Packet Router ensures that output spikes generated by the Neurons in this Core can be routed to their destination whether that is back to another input Axon in the same Core or elsewhere in the Network on Chip (NoC). Basic operation is illustrated in Algorithm 2.1. When a Neuron produces a spike, the destination for this spike is retrieved from the Core SRAM and used to construct a packet for routing through the mesh. A packet's destination is encoded as an offset relative to the producer Core, and it follows a dimension-order/"XY" routing convention where north and east are positive while west and south are negative. Each packet consists of four key fields: signed relative 9-bit *dx* and 9-bit *dy* offsets to route the packet to its destination Core, an 8-bit Axon index to identify the destination Axon (26 bits Spike Destination as shown in 2.3), and a 4-bit delivery tick offset to identify which *tick*, relative to the current time, the Scheduler should schedule this spike for delivery. These last two fields are explored in more detail in discussions in Section 2.3.5.

Figure 2.10 shows the RANC Router architecture. Each Router has Forward East, Forward West, Forward North, Forward South, From Local, and To Local modules, which are used to communicate with both the internal modules of the Core as well as the adjacent cores in every cardinal direction. Forward East, Forward West, Forward North, and Forward South each have one to three FIFO buffers which are necessary for two reasons. First, when mapping a non-trivial application to TrueNorth, the number of packets simultaneously traveling through the network on-chip can be significantly large. At times of high congestion, the buffers are necessary to achieve high throughput. Second, as packets travel through the routing network, the FIFO buffers that capture packets within each forward module become full. This is addressed by applying a form of back-pressure into the network as discussed by F. Akopyan *et al* [41]. Buffers enable back-pressure to ensure that packets are not lost when traveling through the network. All the buffers in RANC Router have a depth is 4 packets.

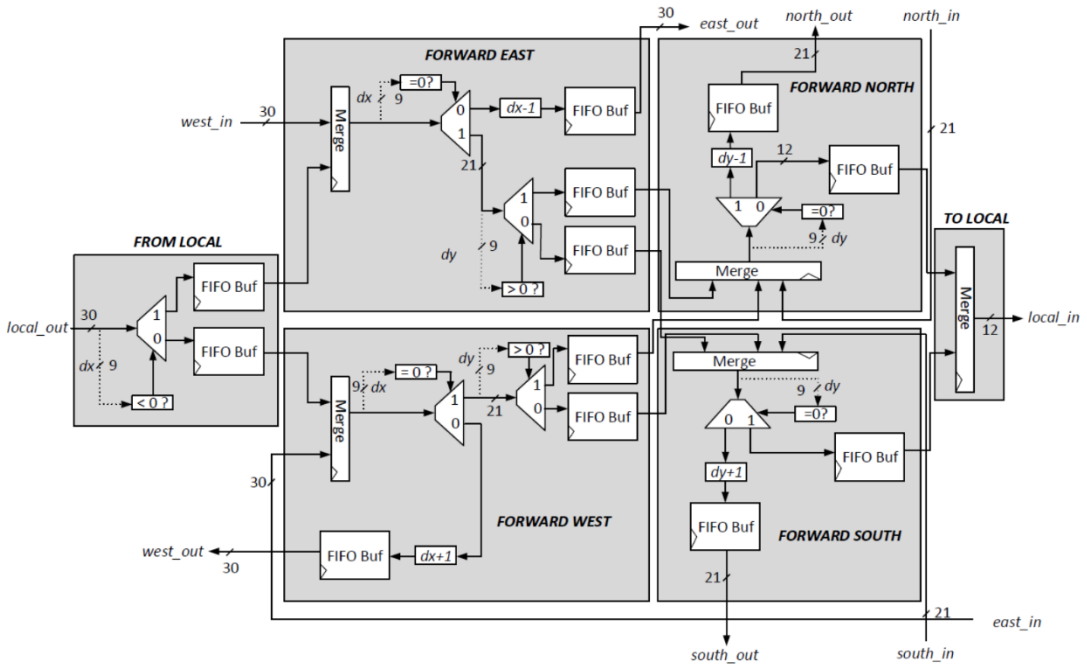


Figure 2.10 RANC Packet Router

2.3.5 Packet Scheduler

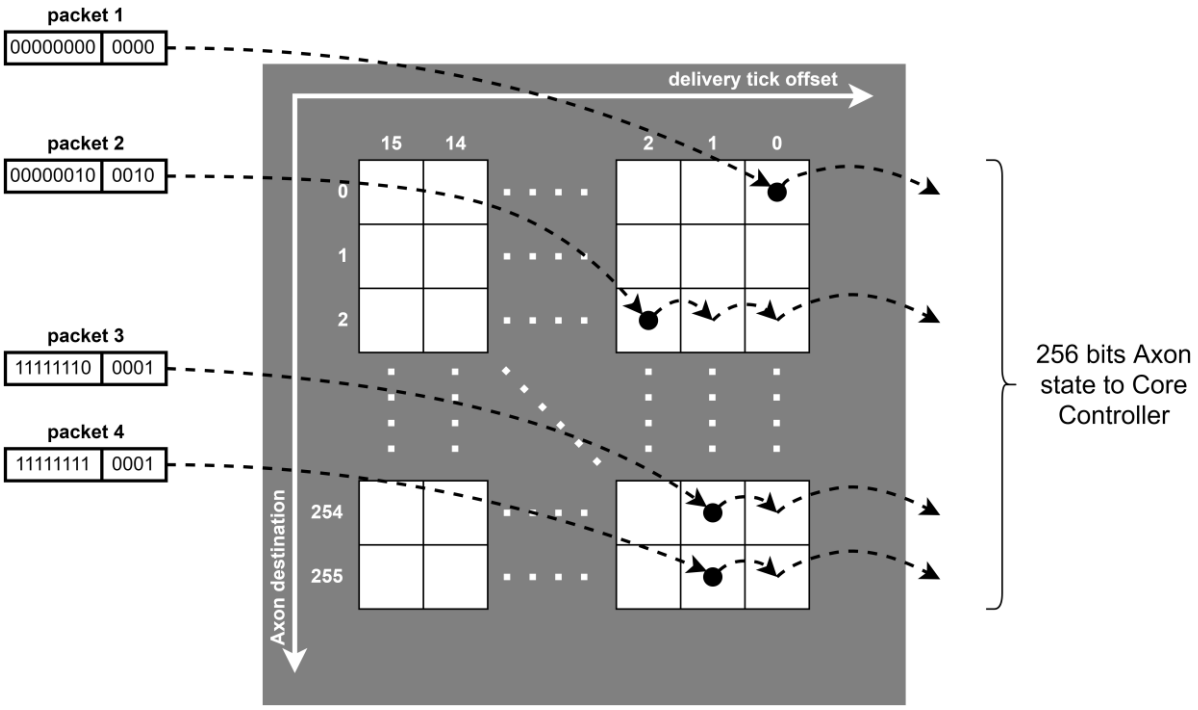


Figure 2.11 RANC Packet Scheduler

The scheduler is the final stop for a spike packet that is traversing the Cores through the routing network. Figure 2.11 shows how Packet Scheduler works. By the time a spike packet arrives at its destination Core, it is reduced to 12 bits with 8 bits of Axon

destination and 4 bits of tick offset values. The Scheduler contains a 256 row by 16 column SRAM. The 8 bit axon value of the spike packet is used to determine which of the 256 rows the newly arrived spike will be written on. The 4 bit tick offset is used to determine which column, with respect to the currently active column, the spike is written to. For example, packet 3 with the value 111111100001 will be transmitted in row 254 (11111110) column 14 (001) of the Scheduler SRAM, after 2 ticks, the spike from this packet will be sent to the Core Controller. In the event that the tick offset causes the spike to be written to the active column, or the current *tick*, an error is thrown and the packet is dropped. This error does not cause any part of the TrueNorth Core to halt. It is used to alert the user that information may no longer be accurate [41].

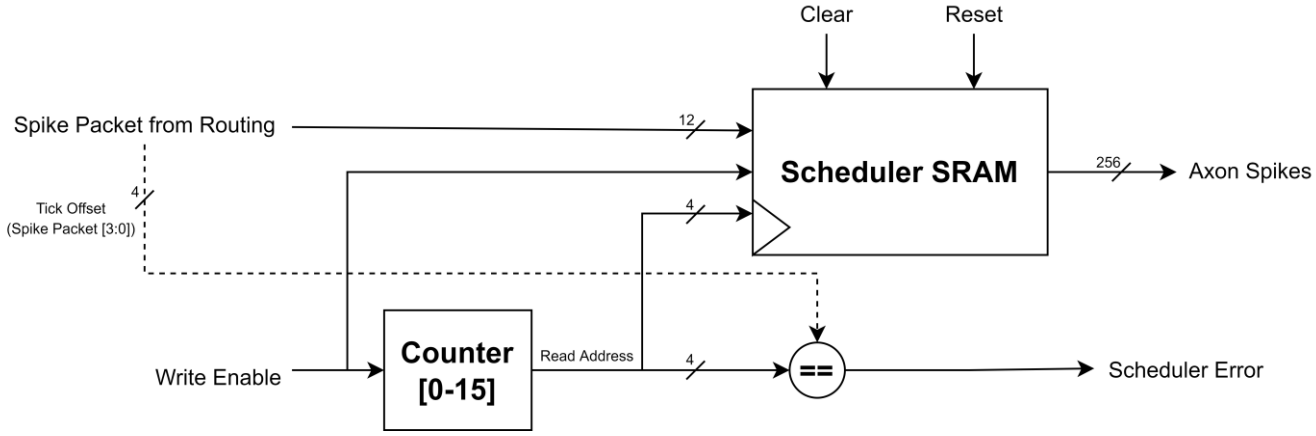


Figure 2.12 RANC Scheduler architecture

The Scheduler is comprised of the Scheduler SRAM block and the 4-bit Counter blocks. The SRAM memory is used to store the spike packets that arrive from the Core's Router. The 4-bit counter increments every time it receives the *tick* as illustrated in Figure 2.12. As the Counter updates based on a signal received from the Core Controller, the Counter value is sent into the Scheduler SRAM as address to send suitable spike to the Neuron Block. This Scheduler does not allow spike packets to be written to a SRAM column corresponding to the current *tick* instance and its also has an error flag to indicate whether packets is suitable for this *tick* or not.

2.4 Conclusion

CHAPTER 2 detailed how the RANC hardware architecture works, which accurately emulates the TrueNorth architecture's behavior with changes compared to the original architecture. Conspicuously in the RANC architecture, the neurons in a Core operate

sequentially, which in turn causes some limitations in processing power. Therefore, a fully parallel core architecture will be presented in CHAPTER 3 to optimize the network's speed and performance.

CHAPTER 3. SNN FULLY PARALLEL ARCHITECTURE

The original architecture of TrueNorth and RANC works very well, but consumes a lot of time and energy, so Vu *et al*. proposed an advanced architecture called SNN Fully Parallel [43], in which all the Neurons in the Core are computed concurrently, instead of operating sequentially as in the original architecture.

3.1 Idea overview

For the RANC architecture, the Neurons are calculated completely independently, without affecting each other's results. However, this computation is performed sequentially, one after the other, among the Neurons. That is, at a time, only one Neuron appears; until this Neuron completes the calculation process, the next Neuron is loaded into the Neuron Block to continue the calculation.

The process of operation of each Neuron is similar when they have to go through 256 Axons to receive stimulation from these Axons (if any). Stemming from this similarity, SNN Fully Parallel architecture is proposed, aiming to perform computations for all 256 Neurons simultaneously. Thus, the 256 Axons traversal will be performed in parallel between the Neurons in a Core. The process of accumulating Membrane Potential for Neurons will only take 256 instead of 256×256 clock cycles, which is a breakthrough in speed compared to the original architecture. Figure 3.1 shows how a core in SNN Fully Parallel architecture behaves when controlled by Token Controller.

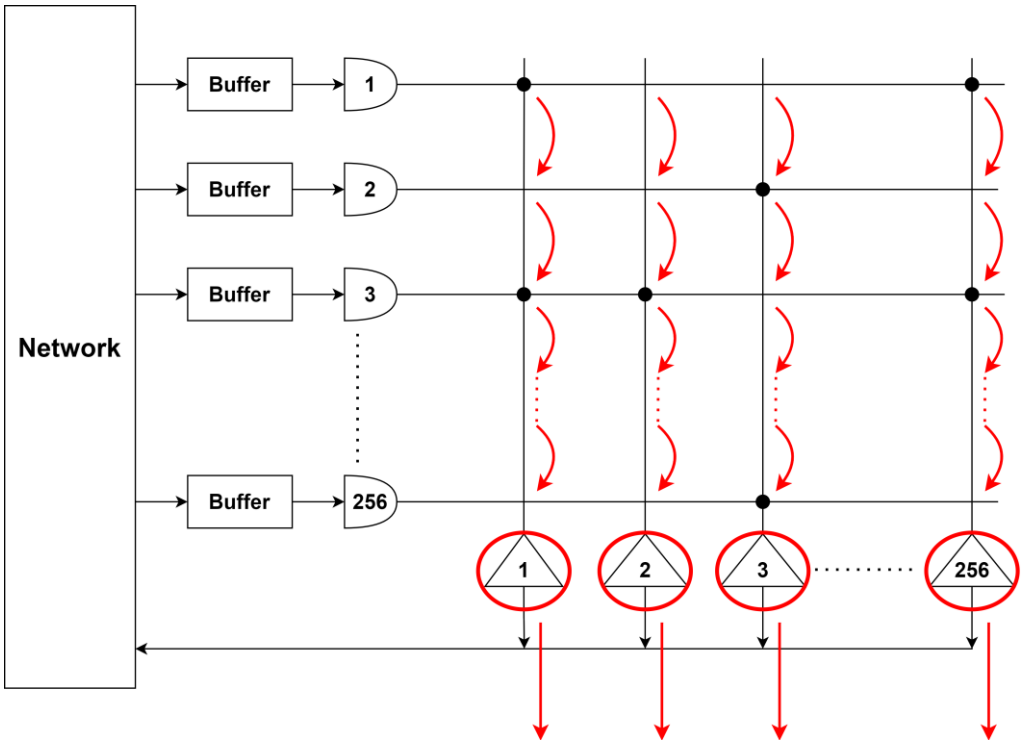


Figure 3.1 SNN Fully Parallel's Core operating principle

In a large SNN including many Cores, the Cores themselves are operated in parallel. So when the computational elements inside the Core, which are Neurons, also work in parallel, we will get a large network of countless Neurons operating at the same time, more accurately describing the structure of the human brain.

3.2 Changes in Core architecture

To implement the SNN Fully Parallel architecture, the idea of loading Neuron data from CSRAM into the Neuron Block "shell" controlled by the Token Controller will be removed. Instead, the Neurons Grid module was formed, with all 256 Neurons appearing simultaneously with all the necessary parameters. Therefore, the general architecture will be changed, each Core will replace three blocks of CSRAM, Neuron Block, and Token Controller into a Neurons Grid block.

3.2.1 New Core architecture

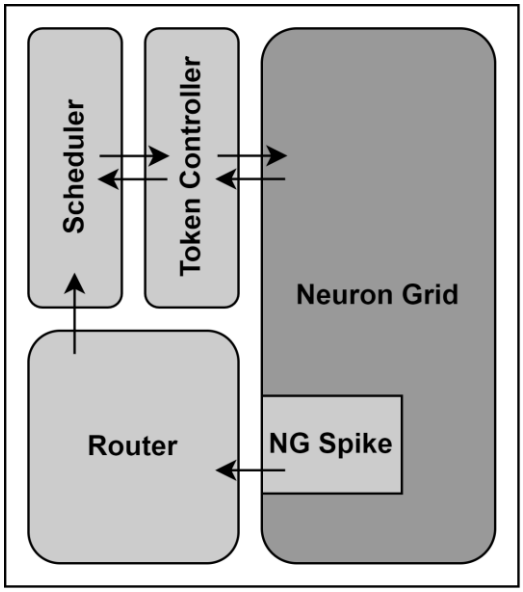


Figure 3.2 SNN Fully Parallel's Core architecture

Figure 3.2 depicts the architecture of a Core for a SNN Fully Parallel SNN architecture. In this new architecture, Core only has 4 main blocks: Router, Scheduler, Token Controller and Neuron Grid. Compared to the TrueNorth's architecture, the two blocks CSRAM, and Neuron Block have been removed and replaced by the Neuron Grid module.

3.2.2 Neuron Grid

The Neuron Grid contains a total of 256 reference samples of the Neuron Block, corresponding to 256 Neurons on the circuit. These Neurons are pre-loaded with the set of parameters obtained during the training of the network. Thus, it can be understood that Neurons Grid is obtained when CSRAM is merged with the 256 Neuron Blocks, resulting in 256 Neurons working at the same time, integrating Membrane Potential and (possibly) release spikes.

However, working in parallel leads to an important consequence that needs to be overcome. For the initial architecture, each Neuron needs to perform computations with 256 Axons, which means it takes 256 clock cycles to be able to deliver a stimulus packet to the Router, and then to the next Neuron to work. Thus, the closest distance between two packets being sent to the Router is 256 cycles. But with the parallel architecture, after traversing to the last Axon of 256 Axons, all 256 Neurons can simultaneously

release the stimulus at the same time, posing a big challenge in batch delivery of packets (up to 256 packets) to the Router appropriately.

Therefore, the Neuron Grid architecture will include a subblock called NG_Spike, and the operation of the Neurons Grid is described in detail as follows (corresponding to Figure 3.3):

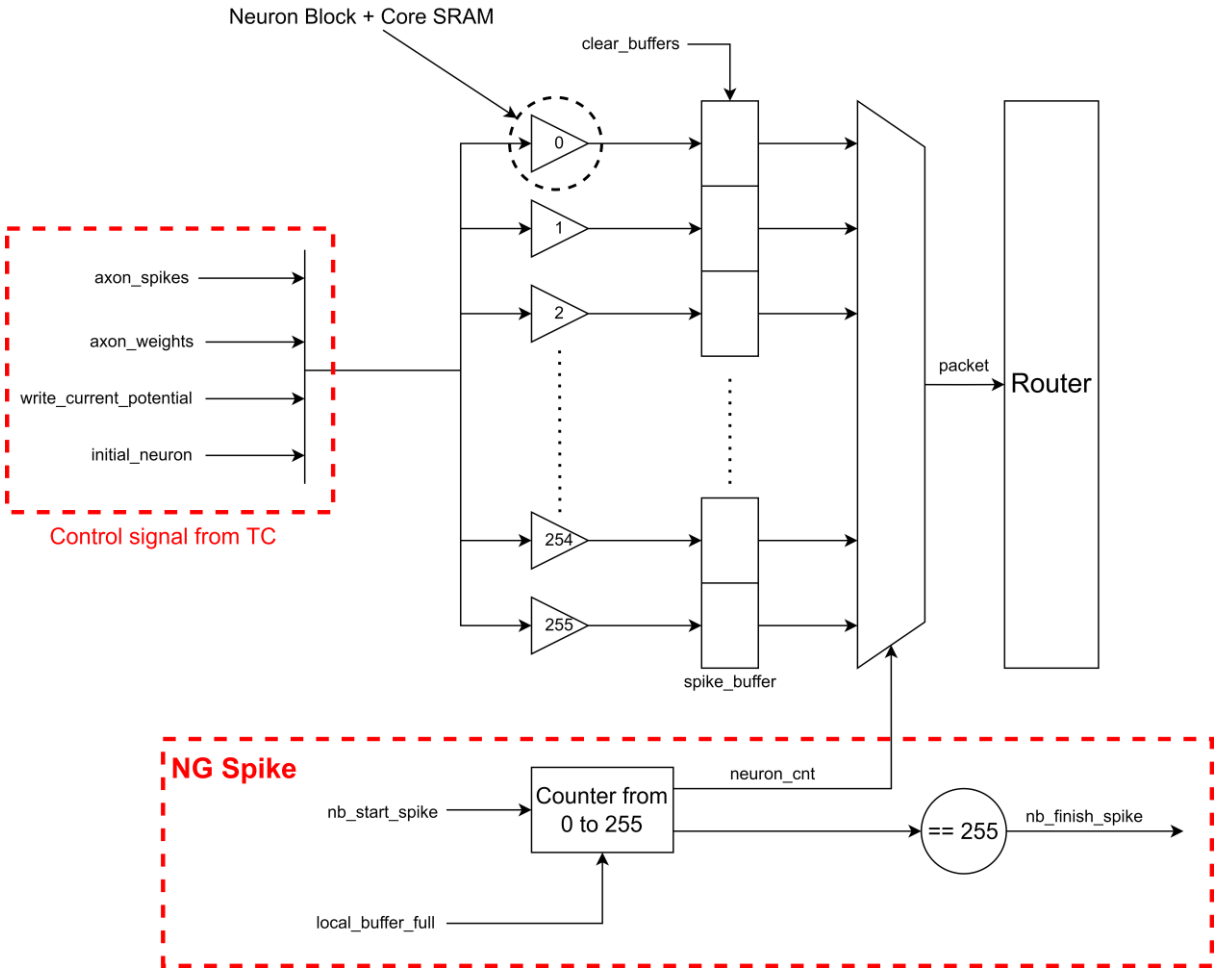


Figure 3.3 Neuron Grid architecture

- 256 Neuron Block of Neurons Grid receives incoming control signals from Token Controller for 256 continuous clock cycles, calculates and integrate Membrane Potential according to 256 Axons.
- When traverse to the last Axons, Neurons whose Membrane Potential value exceeds the positive threshold will simultaneously release the spike to a buffer called *spike_buffer*. Thus, this buffer will store 256 bits corresponding to 256 Neurons, at the n^{th} bit position appears a bit 1 means that the n^{th} Neuron has excitation, and vice versa with bit 0.

- After collecting enough spike into *spike_buffer*, the NG_Spike block waits for the trigger signal to come from the Token Controller. Upon receiving this trigger signal, NG_Spike will in turn deliver the spike packets stored in *spike_buffer* to the Router within 256 clock cycles (if the buffer at the Router is not full).
- After completing the delivery of packets from *spike_buffer* to the Router, NG_Spike gives a finish signal *nb_finish_spike* to the Token Controller signaling the completion of the excitation process into the network.

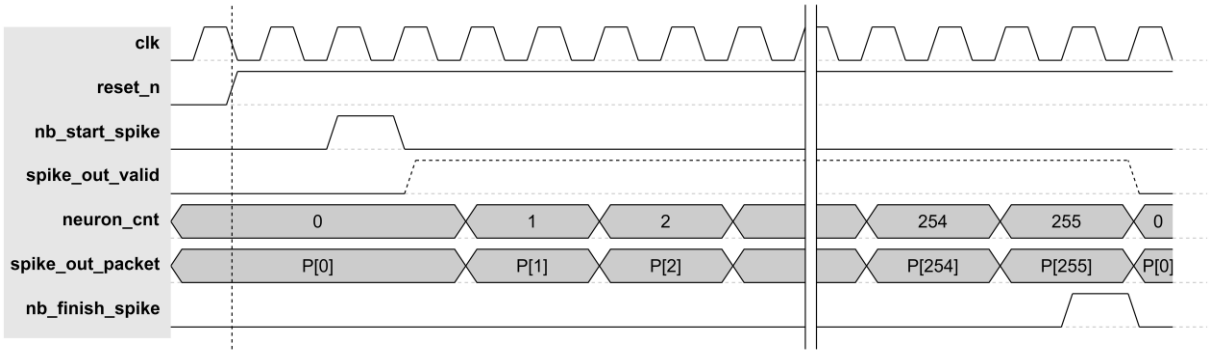


Figure 3.4 SNN Fully Parallel's NG_Spike waveform diagram

Figure 3.4 shows the waveform diagram of NG_Spike block.

3.2.3 Redesign Token Controller

For SNN Fully Parallel network architecture, the communication and control activities between blocks in the network have been changed almost completely. Therefore, the Token Controller block is also redesigned to control parallel operation for 256 Neurons and control the process of sending packets from the Neurons Grid into the network.

After being appropriately redesigned and removing some redundant states that were not really needed in the original architecture, the Token Controller state machine now has a total of 4 states instead of 7 states, the Mealy design style as shown in Figure 3.5 & Figure 3.7; and it will behave as Figure 3.6.

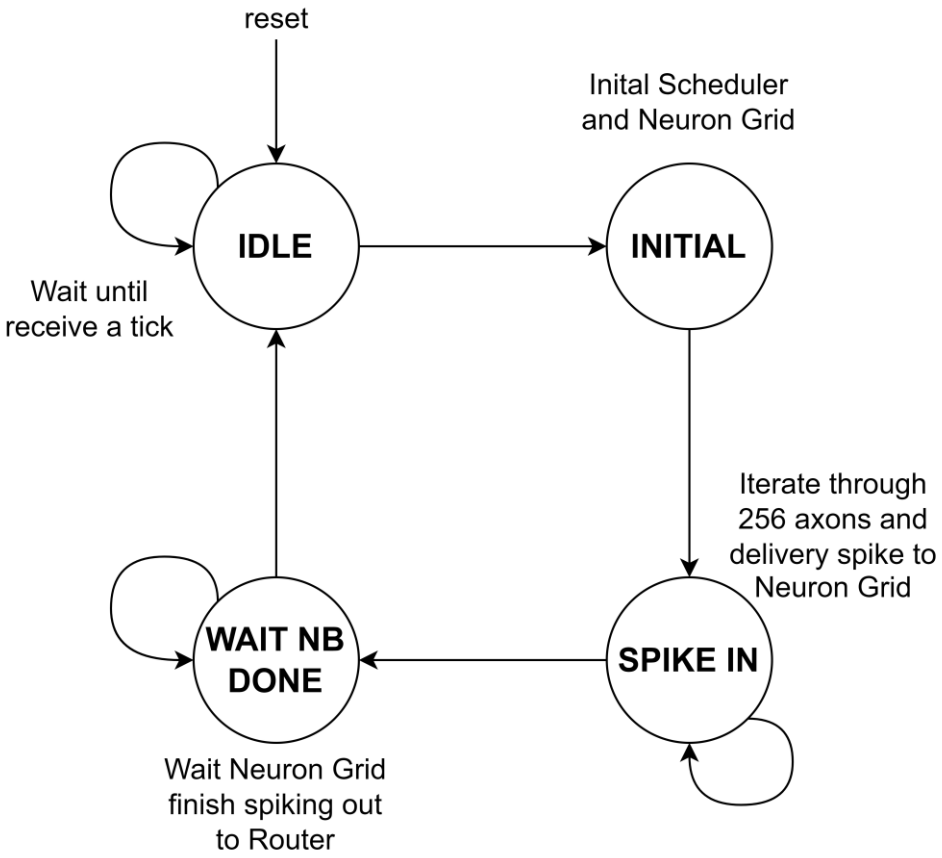


Figure 3.5 SNN Fully Parallel's Token Controller state machine diagram

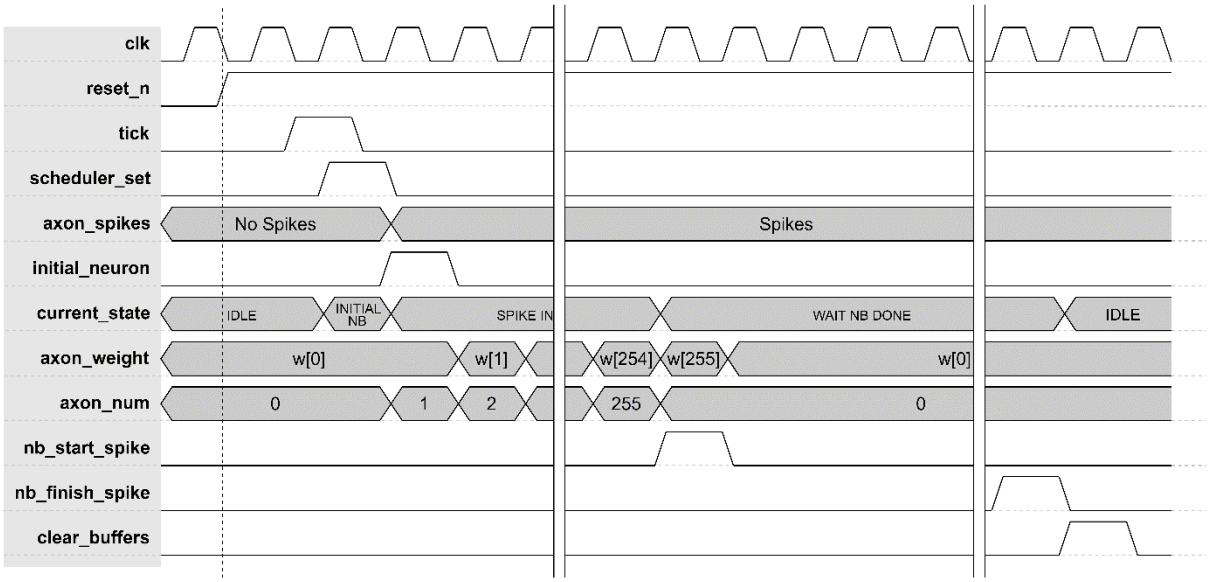


Figure 3.6 SNN Fully Parallel's Token Controller waveform diagram

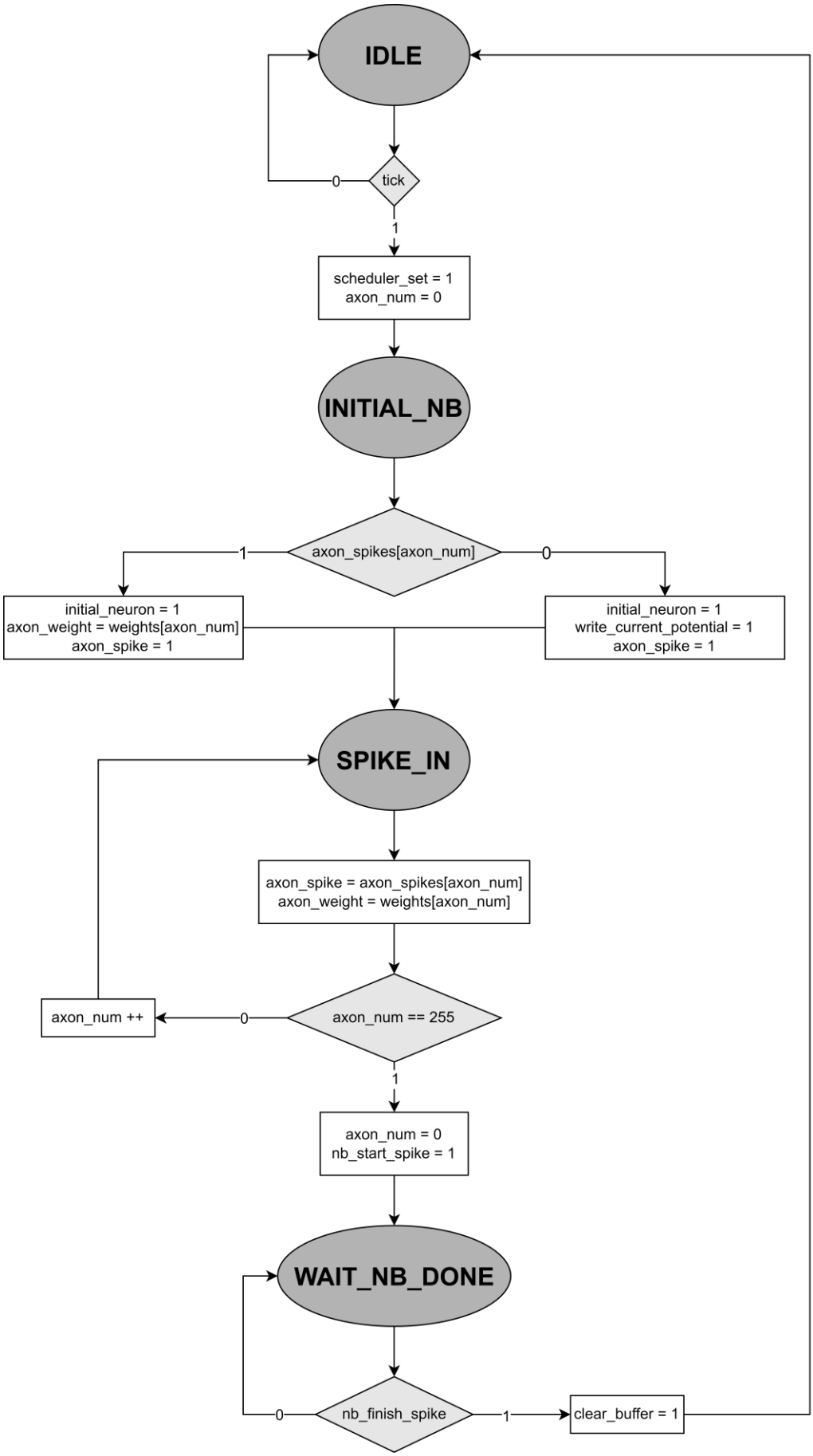


Figure 3.7 SNN Fully Parallel's Token Controller ASMD diagram

When a *tick* is received, the Token Controller sends the *scheduler_set* signal to activate the Scheduler block, which in turn sends control signals to the Neurons Grid to initiate the 256 Axons traversal process. At the position where the n^{th} Axon has spike, the *Axon_spike* signal is raised to 1, and the weight of that n^{th} Axon is also given to the Neuron Grid to calculate for the Neurons.

After finishing traversing to the last Axon, the Token Controller waits for the finish signal (*nb_finish_spike*) from the Neuron Grid to know that this block has released all the spike packets into the network, thereby returning to the original IDLE state, giving a signal to delete the buffers if necessary.

Since the packets are fired from the Cores continuously in clock cycles, congestion is inevitable. Thus, the computation time ($t_{computation}$) for a Core in Fully Parallel SNN architecture can be calculated as the total calculation time ($t_{calculation}$) combined with the total packet propagation time ($t_{propagation}$) to its destination:

- Calculation time equals 1 clock cycle to transform from IDLE to INITIAL_NB; plus 1 more to go to SPIKE_IN; plus 256×2 cycles to iterate through 256 Axon; plus 1 clock cycle to go to WAIT_NB_DONE
  $\Rightarrow t_{calculation} = 1 + 1 + 256 \times 2 + 1 = 515$ (clock cycles)
- Propagation time includes the time to delivery all packets (this time is greater than or equal to 256 clock cycles) and the time for these packets to propagate in the network to reach their destination (unspecified); after finishing deliverying all the output packets
  $\Rightarrow t_{propagation} > 256$ (clock cycles)

The totacl computation time is:

$$t_{computation} = t_{calculation} + t_{propagation} > 515 + 256 = 771\ (clock\ cycles) \quad (3.1)$$

Compared with the estimated results (2.4) in section 2.3.2, this is an outperformed value ($771 \ll 66050$)

3.2.4 Increasing Router's buffer size

When the Neurons work in parallel, it means that the spike packets are generated from the Neurons and launched into the network with a denser frequency. Especially in the case that a Core simultaneously receives packets from the Neurons of many other

neighboring Cores, the buffer overflow in the Router will inevitably occur, making the network become congested, packet loss, and causing the network to fail to perform the original function correctly.

In order to overcome this congestion, the buffers in the Router need to increase in size to be able to store more packets. Packets will be mainly formed and delivered to the Router during 256 clock cycles of the NG_Spike block in section 3.2.2, thereby filling up the buffers in the Router. After this process, NG_Spike will stop delivering packets for a certain period of time until the next tick is received. This time is enough for the Router to propagate the packets to the appropriate destinations, freeing up the buffers.

So the crux of the problem will be solved if the buffers in the Router are large enough to load all the spike packets given from NG_Spike in a short time (256 clock cycles). When conducting the actual survey, the buffer size in the Routers increased from 4 to 128 rows, which will ensure stable and non-congested network operation (with each row having a size of 30 bits, equal to the size of a stimulus package).

However, increasing the buffer size in Routers will cause the size of each core to increase and increase the circuit area for the entire network.

3.3 Conclusion

CHAPTER 3 proposed a completely new architecture compared to the sequential processing architecture mentioned in CHAPTER 2. Here, Neurons in the same Core can operate in parallel and independently. Since then, the network's performance has improved significantly, especially in processing speed. In theory, calculations to compare the processing speed between the two architectures are also performed. From there, the parallel architecture can achieve speeds approximately 86 times faster in the best case than the original architecture. At this time, the functionality of the network remains the same.

However, both architectures mentioned in CHAPTER 2 and CHAPTER 3 are still not fully optimized for ASIC implementation.

CHAPTER 4 will describe in detail how we optimize them to be suitable for ASIC implementation.

CHAPTER 4. OPTIMIZED ARCHITECTURES

In our architecture, Neuron Grid is responsible for the parallel iteration and calculation of each Axon with all 256 Neurons, which is equivalent to the operation of three blocks in RANC architecture: Core Controller, Core SRAM, and Neuron Block; and three blocks in SNN Fully Parallel architecture: Neuron Grid, Token Controller, and NG_Spike. The reason for the combination is that we applied Finite State Machine with Datapath (FSMD)/ Algorithmic State Machine with Datapath (ASMD) methodology to our design in order to reduce the area and increase the speed of the designs compared to the traditional technique [44].

4.1 New Core architecture

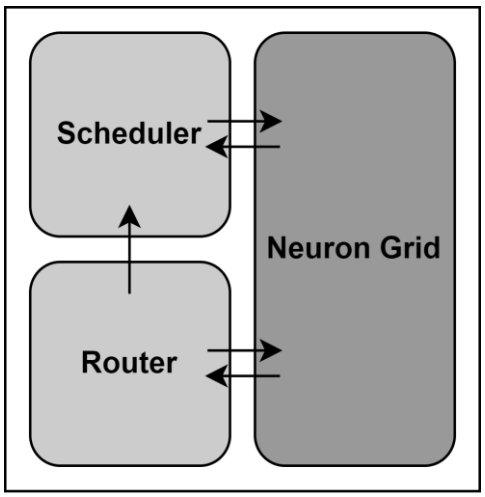


Figure 4.1 New Core architecture

Figure 4.1 depicts the architecture of a Core for our optimized architectures. In this architecture, Core only has 3 main blocks: Router, Scheduler, and Neuron Grid. Compared to TrueNorth's architecture, the three blocks CSRAM, Neuron Block, and Token Controller have been removed and replaced by the Neuron Grid module.

Because our architecture is optimal from the operating principle of both RANC and SNN Fully Parallel architectures, we will temporarily call them SNN Sequential and SNN Parallel, respectively, for brevity.

4.2 Neuron Grid

Since our Neuron Grid is designed based on ASMD/FSMD technique, so it contains 2 main parts. This section will describe in detail how we designed this architecture.

4.2.1 TOP module

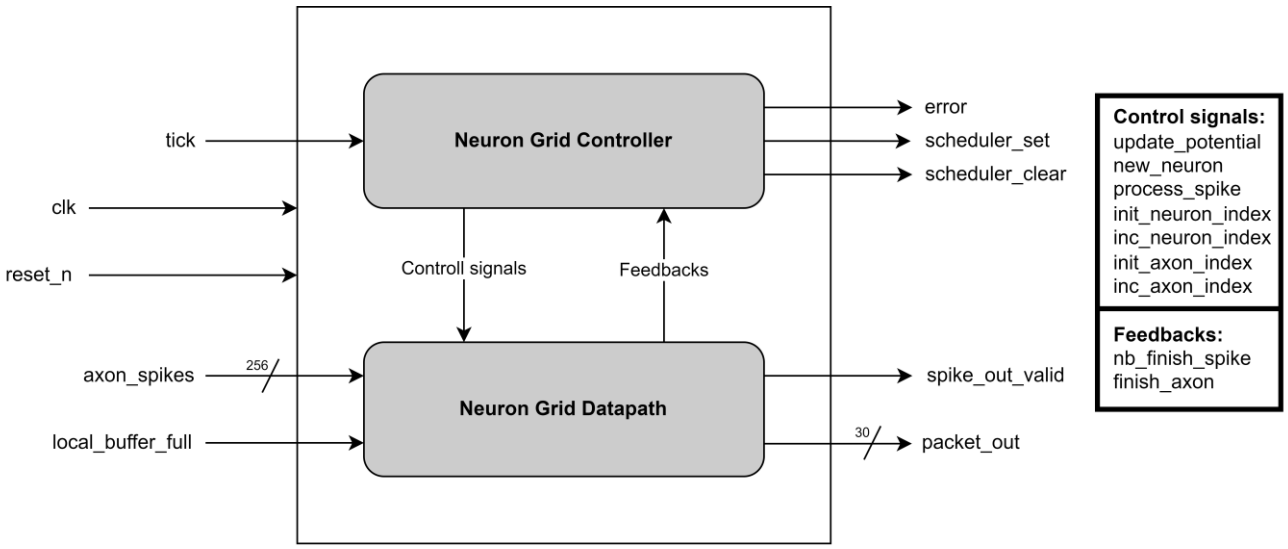


Figure 4.2 Neuron Grid TOP module

Figure 4.2 shows the TOP module of our Neuron Grid.

4.2.2 Interface signals

a) SNN Sequential

Table 4.1 SNN Sequential's interface signals

Signal name	Width	I/O	Description
clk	1	Input	DTI Clock Signal
reset_n	1	Input	DTI Asynchronous Reset, active LOW
tick	1	Input	Start working, active HIGH
local_buffer_full	1	Input	Indicate the moment that the local buffer is full
axon_spikes	256	Input	Axons state information
packet_out	30	Output	Output packet
spike_out_valid	1	Output	Indicate the moment that there is a new packet
sheduler_set	1	Output	Request Axon_spikes from the storage
scheduler_clr	1	Output	Clear the information after taken
error	1	Output	Indicate there is error
update_potential	1	Control	Update new Neuron potential after calculation
new_neuron	1	Control	Prepare for the calculation
process_spike	1	Control	Enable signal for Neuron Block
init_neuron_index	1	Control	Initial Neuron index

inc_neuron_index	1	Control	Increase Neuron index
init_axon_index	1	Control	Initial Axon index
inc_axon_index	1	Control	Increase Axon index
finish_axon	1	Feedback	Indicate that all Axon have been calculated
nb_finish_spike	1	Feedback	Indicate that all Neuron have have been calculated

b) SNN Parallel

Table 4.2 SNN Parallel's interface signals

Signal name	Width	I/O	Description
clk	1	Input	DTI Clock Signal
reset_n	1	Input	DTI Asynchronous Reset, active LOW
tick	1	Input	Start working, active HIGH
local_buffer_full	1	Input	Indicate the moment that the local buffer is full
axon_spikes	256	Input	Axons state information
packet_out	30	Output	Output packet
spike_out_valid	1	Output	Indicate the moment that there is a new packet
sheduler_set	1	Output	Request Axon_spikes from the storage
scheduler_clr	1	Output	Clear the information after taken
error	1	Output	Indicate there is error
update_potential	1	Control	Update new Neuron potential after calculation
new_neuron	1	Control	Prepare for the calculation
process_spike	1	Control	Enable signal for Neuron Block
init_neuron_index	1	Control	Prepare for the packets delivery
inc_neuron_index	1	Control	Change address in packet buffer
init_axon_index	1	Control	Initial Axon index
inc_axon_index	1	Control	Increase Axon index
finish_axon	1	Feedback	Indicate that all Axon have been calculated
nb_finish_spike	1	Feedback	Indicate that all packets have been delivered

4.2.3 Functional description

a) SNN Sequential

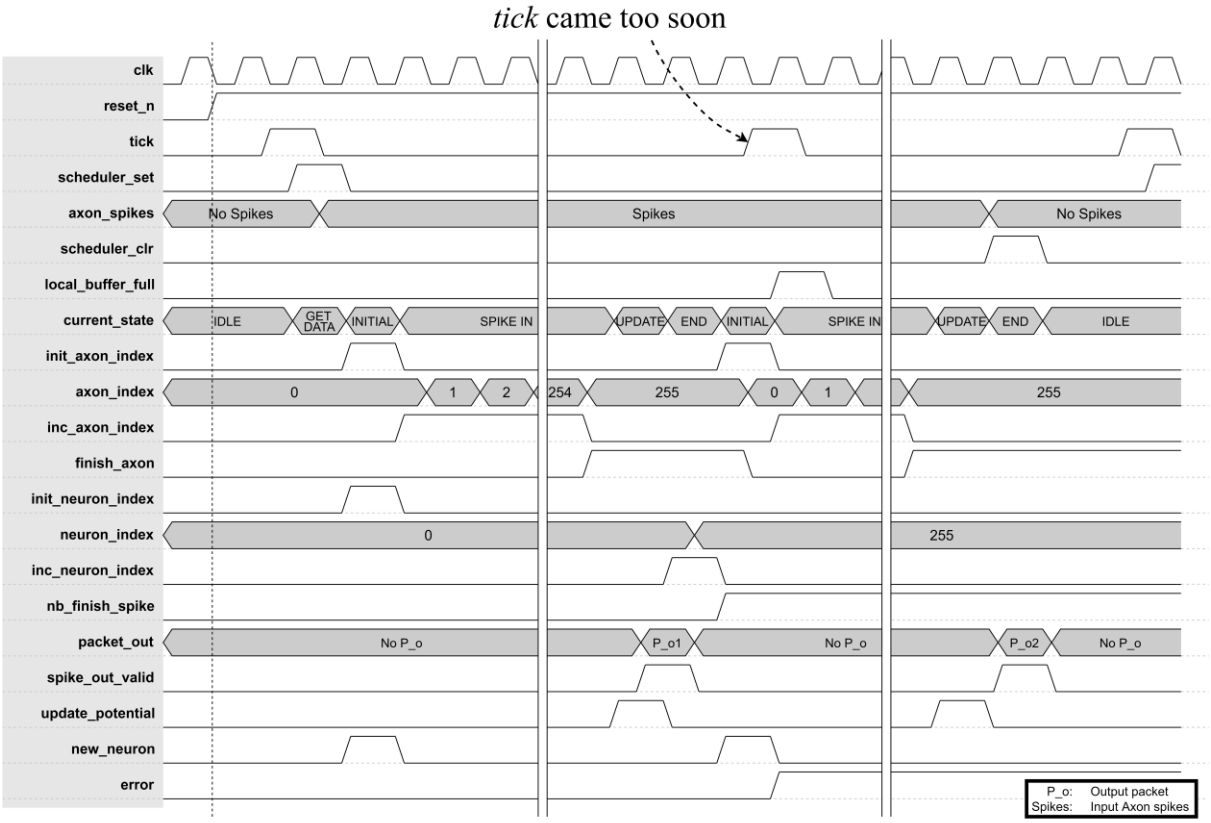


Figure 4.3 SNN Sequential's Neuron Grid waveform diagram

Figure 4.3 describes how our new design behaves.Figure 4.3 SNN Sequential's Neuron Grid waveform diagram

Different from the original RANC architecture, the Core Controller, Core SRAM, and Neuron Block blocks are combined to form a single Neuron Grid architecture. The Controller of this new architecture acts as the Core Controller, and the Datapath will include all 256 Neurons data stored in flip flop arrays.

Table 4.1 shows all the interface signals in our module.

The working principle of this new Neuron Grid will be similar to the original architecture. Specifically, when receiving the *tick*, the Controller will in turn switch states to proceed with the request (*scheduler_set*) to receive 256 bits of input *axon_spikes* and calculate.

It will perform calculations for each Neuron one by one, and each Neuron will calculate for each Axon one by one. If there is a spike at that Neuron, the Neuron Grid will send a signalling signal (*spike_out_valid*) and proceed to send the packet out (*packet_out*). The process of sending the packet out will depend on the congestion in the network signed by *local_buffer_full*. The whole computation will be 256×256 loops.

After the calculation is complete, the Neuron Grid will send the *scheduler_clr* to the Scheduler to delete the old axon states, wait for a new *tick* to be sent, and repeat the process.

If the calculation is in progress, a *tick* arrives then an *error* flag will be raised.

b) SNN Parallel

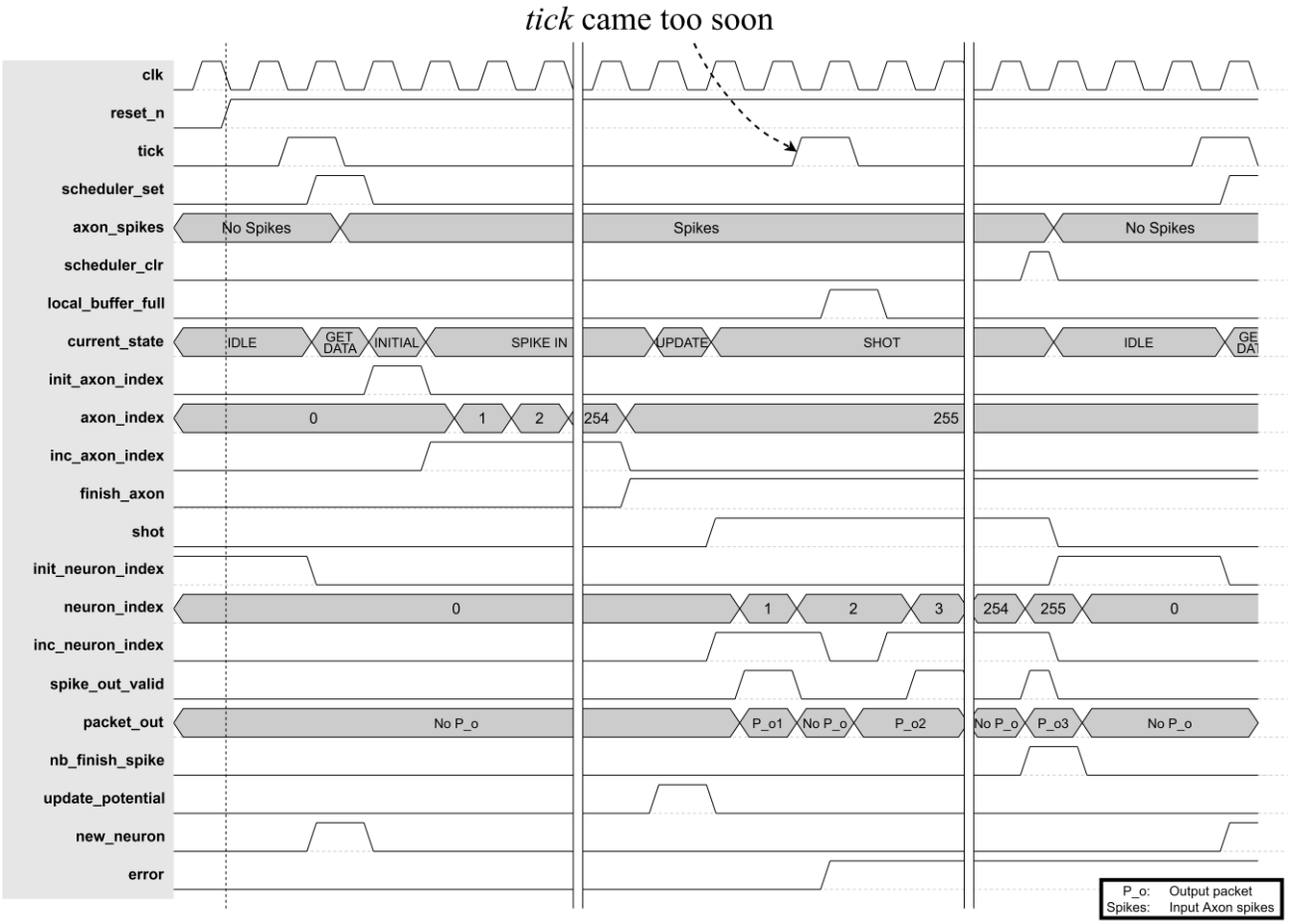


Figure 4.4 SNN Parallel's Neuron Grid waveform diagram

Our Neuron Grid is the combination of Neuron Grid and Token Controller of SNN Fully Parallel architecture, so the block has some connections with Scheduler and Router. When The Controller receives a *tick*, it will send the Scheduler a few signals

(*scheduler_set* and *scheduler_clr*) for setting and clearing. The Datapath receives buffer full announcement (*local_buffer_full*) from Router and Axon state (*axon_spikes*) from the Scheduler, which is stripped from Core's input packets. On the contrary, Packets (*packet_out*) and valid (*spike_out_valid*) signals for them are transmitted to Router by the Datapath. If the *tick* signal occurs before computation completes (i.e., f_{tick} is too fast), an *error* flag is raised to notify the user that the output may be corrupted.

For this new architecture, we have integrated the NG_Spike block into the Neuron Grid. Specifically, when the Neuron Grid finishes computing for all 256 Neurons, the Controller will send a *shot* signal to the Datapath, which will combine with *local_buffer_full*, and *neuron_index* to send packets out sequentially according to clock pulses.

All of the above signals were shown in Table 4.2, and they will behaves as Figure 4.4.

4.2.4 Architecture

4.2.4.1 SNN Sequential

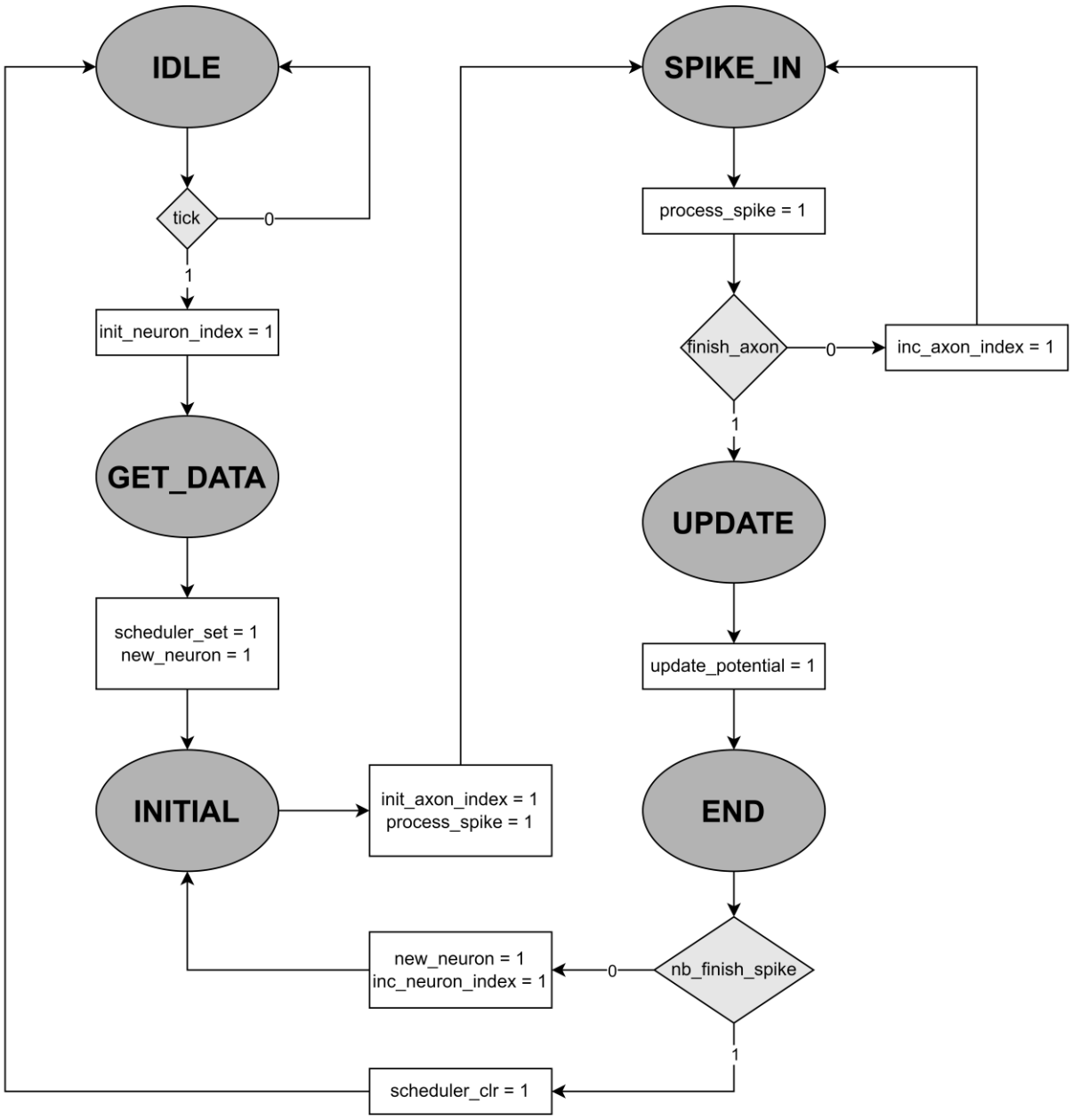


Figure 4.5 SNN Sequential's Neuron Grid Controller ASMD diagram

There are not too many changes compared to the original architecture (CHAPTER 2) in terms of operating principles, we just redesigned the controller in a more optimal way. Figure 4.5 shows the ASMD diagram of this new architecture.

4.2.4.2 SNN Parallel

a) SNN Parallel's Neuron Grid Controller

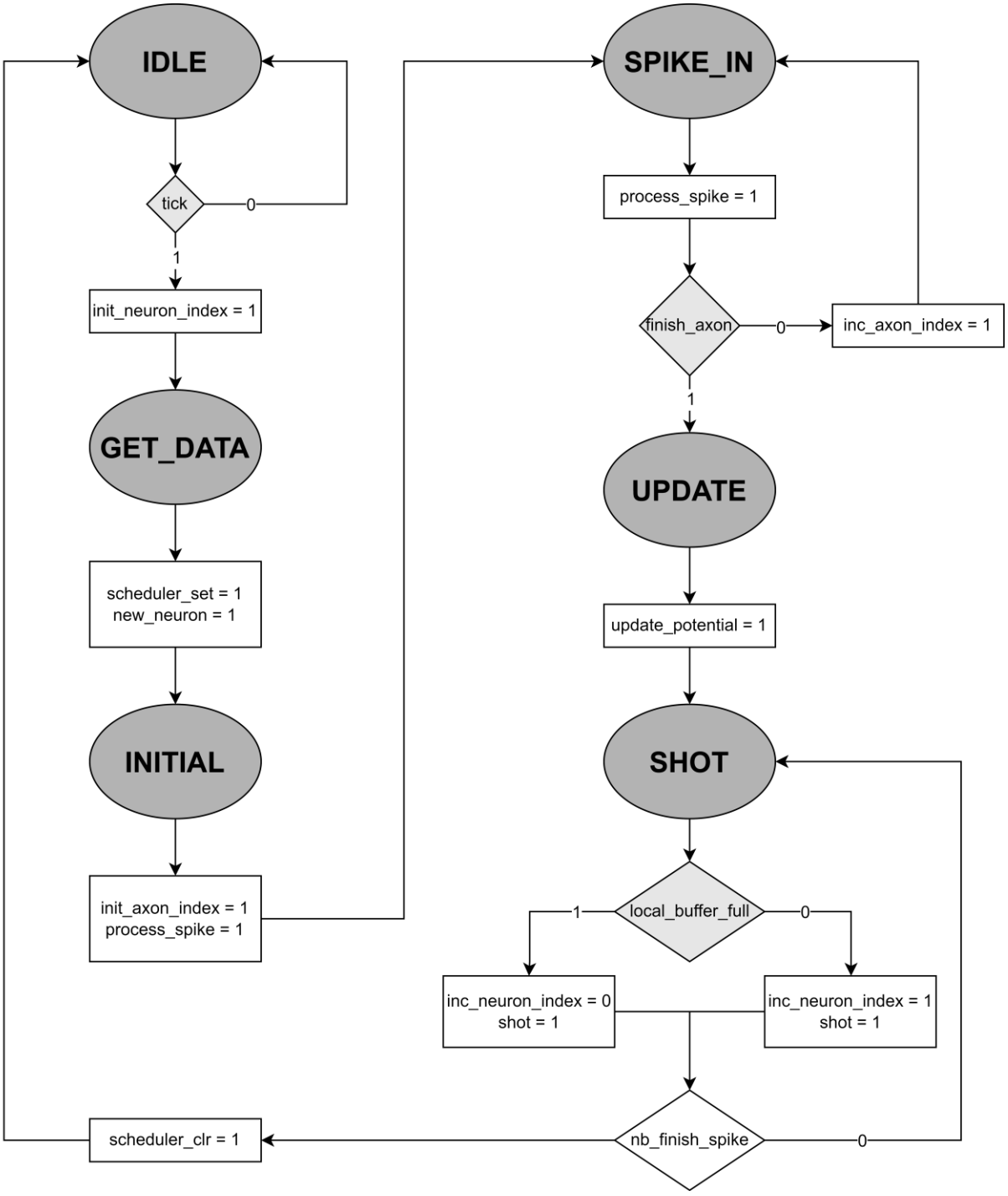


Figure 4.6 SNN Parallel's Neuron Grid Controller ASMD diagram

In SNN Fully Parallel core, to avoid buffer overflow, spike packets are firstly stored in a spike buffer, then consecutively delivered to Router by NG_Spike block. However, our design takes advantage of the Controller to serially send the packets to Router by increasing the indexes in a 256-bit buffer storing generated spikes in the Datapath.

Figure 4.6 shows how our Controller behaves when receiving a *tick*. Thus, in IDLE state, *init_neuron_index* is sent by the Controller to reset the index of the buffer (Packet Buffer) in the Datapath. In the same state, the Controller waits for a clock cycle to send a request to Scheduler (*scheduler_set*) to send Axons state to Neuron Grid and require the Datapath to load current Membrane Potential of Neurons in one clock cycle. In the next state, the Controller indicates that the Datapath proceeds with the first Axon (*new_neuron*, *init_axon_index*) and sets the *process_spike* signal. The iteration of all 256 Axons is equivalent to 256 clock cycles that the Controller gives the Datapath a command. When all the Axons are processed, the Controller receives feedback (*finish*_axon) from the Datapath and controls the Datapath (*update*_potential) to update the Membrane Potential of the Neurons. All spikes from 256 Neuron Blocks will be stored in a buffer called Packet Buffer, which uses an address variable to control packet output. In State SHOT, the Controller sends *shot* signal to drive the Datapath to consecutively deliver packets to Router. If the FIFO buffer of the Router is not full, it continuously requests the Datapath to transmit all packets by increasing the index inside Packet Buffer (*inc_neuron_index*). When all the packets are successfully sent, the Controller clears the Scheduler and returns to IDLE state.

The computation time of this new Controller can be estimated as follows:

- When receive a *tick*, Neuron Grid Controller takes 3 clock cycles to go through GET_DATA, INITIAL and reach SPIKE IN
- Neuron Grid Controller spends 255 more clock cycles at SPIKE IN state to iterate through all 256 Axons.
- After calculating for all Axons of all the Neurons, it takes 1 clock cycles to get to SHOT.
- The time it stays at SHOT can be more than 256 clock cycles, depending on the network's congestion (unspecify).

⇒ So the total computation time should be:

$$\tau_{computation} = 3 + 255 + 1 + (\geq)256 + 1 \geq 516\ (clock\ cycles) \qquad (4.1)$$

In SNN Fully Parallel architecture, each Axon takes two clock cycles to compute; this is to avoid causing over-congestion in the network. However, this congestion is mainly controlled by the Buffers in the Core's Router, so increasing the computation

time for an Axon from one clock cycle to two clock cycles makes no sense. Therefore, it can be concluded that the estimated results from our architecture are approximate compared with the result (3.1) in section 3.2.3.

b) SNN Parallel's Neuron Grid Datapath

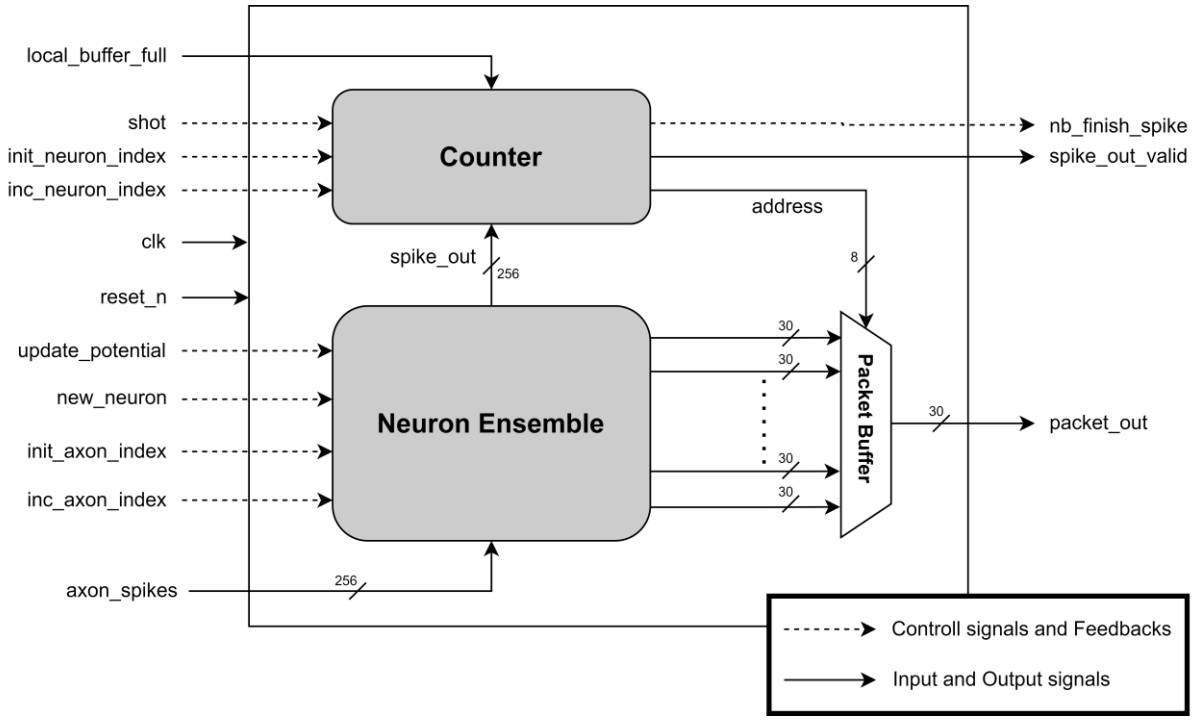


Figure 4.7 SNN Parallel's Neuron Grid Datapath block diagram

In Neuron Grid Datapath, all 256 Neurons are preloaded with trained parameters in many flipflops so that they can operate parallelly. Those Neuron Blocks are included in the Neuron Ensemble block (as shown in Figure 4.7), which is responsible for integrating the Membrane Potential of all the Neurons, evaluating if a spike is generated, and sending packets to Router. The Neuron Ensemble block receives control signals from the Controller, which were described above, to load necessary parameters and perform the calculations. After all the Neuron Blocks finish operating, all the Neurons having the Membrane Potential greater than the Threshold generate spikes which are transmitted to the Counter and stored in a buffer. A feedback signal from the Datapath that the last Axon has been proceeded is given to the Controller and the Membrane Potential of the Neurons is then updated by the Datapath based on Controller's request. When receiving Controller's command to send packets in state UPDATE and the "!*local_buffer_full*" signal from the Router, the Counter makes use of generated spike in the buffer to generate the addresses so that appropriate packets are sent to the Router.

In order to consider all the spikes, the index of the buffer which is reset to 0 in IDLE is requested to be increased by the Controller in state 5. When the index reaches 255, Datapath sends a signal to the Controller to report that all the packets are sent.

4.3 Decreasing Router's buffer size

In SNN fully parallel, they proposed the buffer size in the Router increasing from 4 to 128 rows will ensure the network operation is stable and not cause congestion. However, after several simulations and verification were carried out with their proposed FIFO buffer, the results showed that the problem of overflow stills existed. At the same time, the required area for this design was extremely large because of the great number of FIFO buffers. As a result, we decided to reduce the size of the FIFO buffer from 128 to 4 like the RANC to minimize the area without significant alterations in throughput.

4.4 Conclusion

In CHAPTER 4, by using the ASMD/FSMD design approach with some minor modifications, 2 more optimized versions of the architectures mentioned in CHAPTER 2 and CHAPTER 3 were produced, ready for ASIC synthesis.

CHAPTER 5 will describe how to train the SNN to perform the simulation problem in the project.

CHAPTER 5. SNN TRAINING METHODOLOGY

This section will discuss how we train the Neuron parameters by software before loading it to hardware implementation. Specifically, we will fix the assignment of Axon's weights and find the synaptic connections between Neurons and Axons, or the crossbar in Figure 2.2. This crossbar defines how Neurons connect with others inside a core so that it helps solve specific problems, which can be considered as the condition for the software training task.

5.1 ANN overview

5.1.1 Neuron model

ANNs, usually simply called neural networks (NNs) or, more simply yet, neural nets, are computing systems inspired by the biological neural networks that constitute animal brains.

The researchers sought to convert their understanding of how biological Neurons work into computer-operable Artificial Neural Network models. Figure 5.1 shows a model of a single Neuron, seen as the basic information processing unit of a neural network. These Neurons are used to build neural networks with more complex architecture:

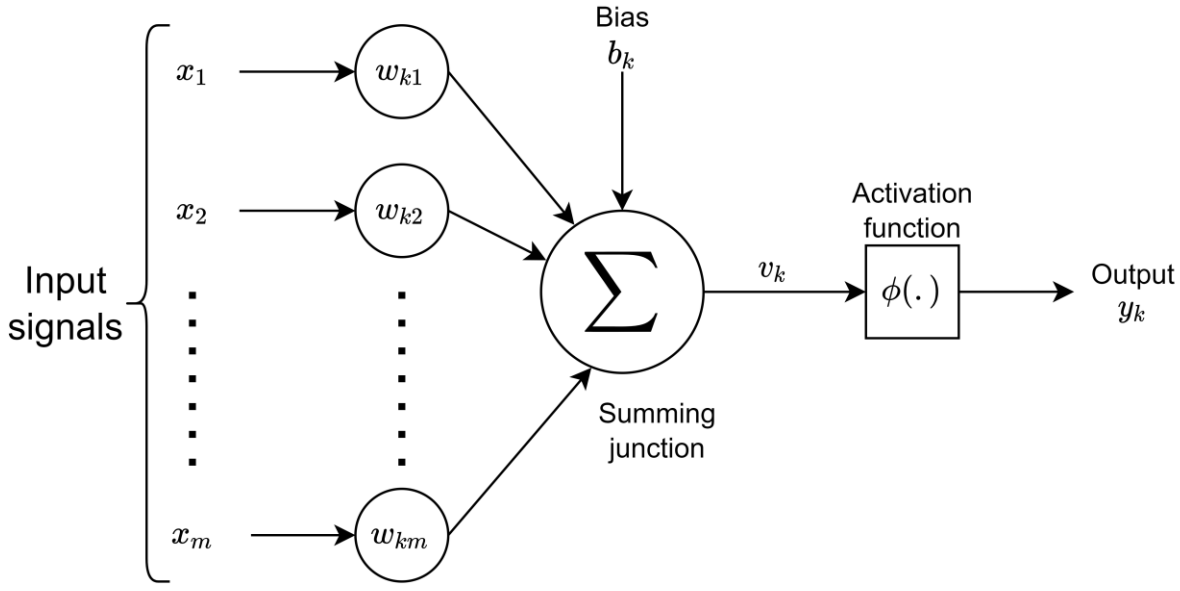


Figure 5.1 Model of an artificial Neuron labeled k

Every artificial Neuron includes:

- A set of *synapses* or also known as *connecting links* used to connect Neurons together. Each synapse is characterized by its link strength. More specifically, a synapse is used to transfer a signal from a Neuron labeled j to a Neuron labeled k with a weight wkj. Unlike the weight of a synapse in a biological nervous system, the weight of an artificial synapse can be either negative or positive.
- An adder is used to aggregate the signal heads into each Neuron and send the result forward.
- An activation function is used to bring the output signals of a Neuron to a certain range of values or a fixed set of values.

We can describe the activity of the Neuron labeled k in the figure above using the following mathematical equations:

$$y_k = \varphi\left(\sum_{i=1}^{m} w_{ki}x_i + b_k\right) \quad (5.1)$$

Which,

m	is the number of inputs to Neuron k
x_i	is the i^{th} input value of Neuron k
w_{ki}	is the weight of Neuron k corresponding to the i^{th} input value
b_k	is the bias value of Neuron k
$\varphi(.)$	is the activation function of Neuron k
y_k	is the output value of Neuron k

5.1.2 FeedForward model

Neural networks are created when we connect single Neurons together. In a neural network, Neurons are organized into layers. In its simplest form, a neural network has only one output layer. The input signals are passed directly through the weights. This simple design lays the foundation for other neural networks with more complex structures.

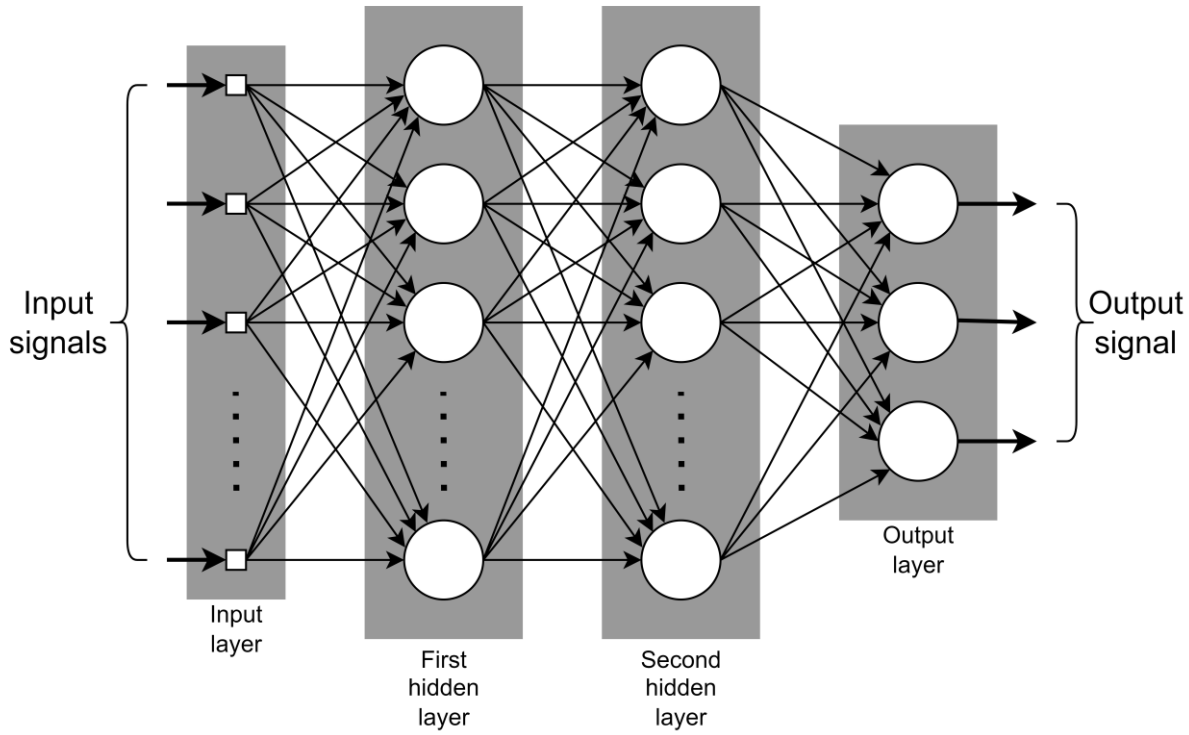


Figure 5.2 FeedForward model with 4 layer

Figure 5.2 shows the architecture of a feedforward neural network, which is very similar to the one proposed by McCulloch and Pitt in 1943. Normally Neurons are arranged in layers and connect to each other. The first layer on the left is the input layer, and the last layer on the right is the output layer, the remaining layers in the middle are the hidden layers. The Neurons in one layer will have a weight corresponding to the connection to the Neuron of the other layer. Each Neuron in any layer of the network is connected to all the Neurons in the previous layer. This type of connection is called fully connected. There are no connections between Neurons in the same layer. Signals are only transmitted in one direction, from left to right, through the Neurons, and none of the connections transmit signals in the opposite direction.

5.1.3 Training methodology

As mentioned by [45], training consists of the selection of coefficients for each Neuron in the layers so that with certain input signals we get the necessary set of output signals.

There are 4 learning methods:

- Supervised learning
- Unsupervised learning

- Semi-Supervised learning
- Reinforcement learning

And 3 main training methods:

- Batch training: All input patterns are fed to the network at the same time and then update the network weights concurrently.
- Online training: The network weights are updated immediately after an input pattern is fed into the network.
- Stohastic training: Same as online training, but the selection of input patterns to feed into the network from the training set is done at random.

In the framework of this project, I will only mention about Supervised learning. In this kind of method, there is a training set (dataset) that contains examples with true values: tags, classes, indicators.

Neural networks are trained in two stages: forward error propagation and reverse error propagation.

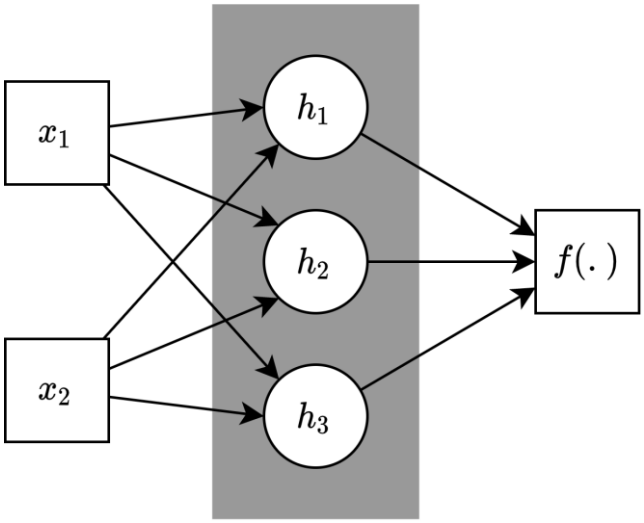


Figure 5.3 Neural network with 2 inputs, 3 Neurons and 1 output

During forward error propagation, a prediction of the response is made. For instance, for neural networks with two inputs, three Neurons, and one output as shown in Figure 5.3, we can set the initial weights for all Neuron at random: w_1, w_2. Multiply the input by the weights to form a hidden layer:

$$h_1 = (x_1 \times w_1) + (x_2 \times w_1)$$
$$h_2 = (x_1 \times w_2) + (x_2 \times w_2) \quad (5.2)$$

$$h_3 = (x_1 \times w_3) + (x_2 \times w_3)$$

The output from the hidden layer is transmitted through a non-linear function (activation function), to obtain the network output:

$$\hat{y} = f(h_1, h_2, h_3) \quad (5.3)$$

With backpropagation, the error between the actual response and the predicted response is minimized. Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single input-output example. This makes it possible to use gradient methods for training multi-layer networks, updating weights to minimize loss. Gradient descent or variants such as stochastic gradient descent are commonly used.

The total error is calculated as the difference between the expected value of y (from the training set) and the obtained value of $\hat{y}$ (calculated at the stage of direct propagation of the error) passing through the cost function. The purpose of the problem is to optimize the loss function

$$Min(\mathcal{L}(y, \hat{y})) \quad (5.4)$$

The partial derivative of the error is calculated for each weight (these partial differentials reflect the contribution of each weight to the total loss.

Then these differentials are multiplied by a number called the learning rate (η). The result is then subtracted from the corresponding weights. The result is the following updated weights:

$$\begin{aligned} w_1 &= w_1 - \left(\eta \times \frac{\partial(\mathcal{L}(y, \hat{y}))}{\partial(w_1)}\right) \\ w_2 &= w_2 - \left(\eta \times \frac{\partial(\mathcal{L}(y, \hat{y}))}{\partial(w_2)}\right) \\ w_3 &= w_3 - \left(\eta \times \frac{\partial(\mathcal{L}(y, \hat{y}))}{\partial(w_3)}\right) \end{aligned} \quad (5.5)$$

The fact that we assume and initialize the weights in a random way, and they give accurate answers, does not sound quite reasonable; however, it works well.

The choice of the starting point for complex neural network architectures is a rather difficult task, but for most cases, there are proven technologies for choosing the initial approximation.

In addition, network training is currently not conducted on the entire data set, but on samples of a certain size, the so-called batches. This means that weights in neural networks are tuned from epoch to epoch, to produce better results.

5.1.4 Outcome

After conducting such a training process, the result will be a neural network connected to each other through branches, each branch is assigned a definite weight. The Neurons communicate with each other using real numbers, in other words the information is now encoded in the magnitude dimension as shown in Figure 5.4.

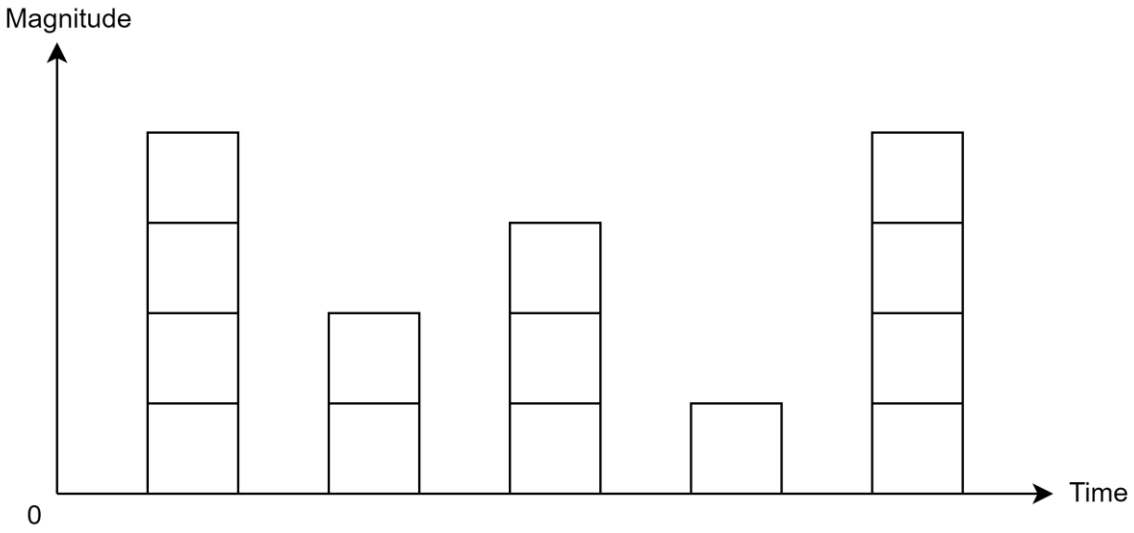


Figure 5.4 Information is encoded in magnitude dimension

5.2 Tea layer

To describe the LIF model, Tea Layer of the RANC project was born. The essence of Tea Layer is the same as Fully connected layer from ANN.

5.2.1 Neuron model

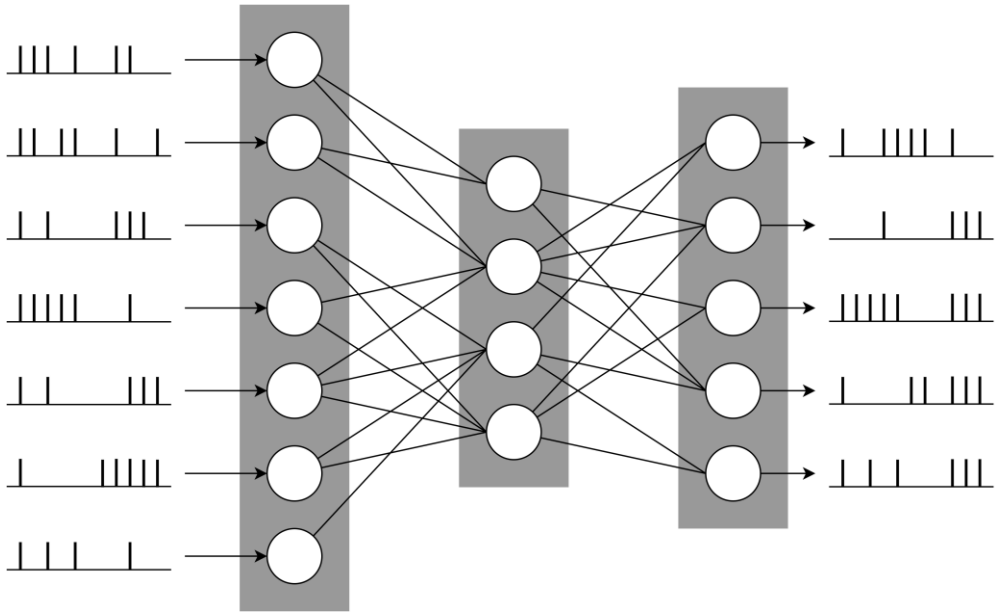


Figure 5.5 FeedForward Spiking neural network model

Figure 5.5 is an example of SNN. Specifically, in an SNN, Neurons communicate with each other using spikes. In this model, each Neuron includes an Axon branch to communicate with other Neurons, and a bunch of Dendrites (up to 256 branches) connected to Axons from other Neurons, this connection point is called a synapse. When a Neuron receives a pulse, the Membrane Potential of this Neuron will be accumulated, and it will gradually accumulate until it exceeds a certain threshold, then proceed to fire a new spike through its Axon to communicate with other Neurons.

Therefore, the output of Neuron k takes only one of two values, 0 (no spike) or 1 (spike) based on that Neuron's Membrane Potential I_k value:

$$n_k = \begin{cases} 1 & If\ I_j \geq 0 \\ 0 & Otherwise \end{cases} \quad (5.6)$$

It can be easily seen that the activation function of the Neuron is now the same as the Heaviside function, but the Heaviside function has no derivative, so backpropagation cannot occur. The solution here is to approximate this Heaviside function. One of the possible solutions is to use the Sigmoid function instead, then the predicted output $\widehat{n_k}$ of Neuron k is calculated as follows:

$$\widehat{n_k} = \frac{1}{1 + e^{-2j \times I_k}} \quad (5.7)$$

Where j is the slope of the function, usually $j = 0.5$ is used. Then the derivative formula is calculated:

$$\frac{\partial \widehat{n_k}}{\partial I_k} = \frac{1}{1 + e^{-2j\,I_k}} \times \left(1 - \frac{1}{1 + e^{-2j\,I_k}}\right) \quad (5.8)$$

5.2.2 Training methodolody

As shown in Figure 2.2, the brain is made up of many Neurons, and RANC assigns these Neurons to Neuron clusters, where each cluster consists of 256 Neurons and 256 Axons input, each Neuron consists of up to 256 Dendrites and 1 Axon. Considering the operation of Neuron k gives the folowing formula:

$$I_k = \sum_{i=0}^{256} x_i a_{ki} c_{ki} + b_k \quad (5.9)$$

Hence:

$$n_k = heaviside(I_k) = heaviside\left(\sum_{i=0}^{256} x_i a_{ki} c_{ki} + b_k\right) \quad (5.10)$$

Which,

x_i is the input spike at Axon i

a_{ki} is the weight of Axon i belonged to Neuron k

c_{ki} is the connection of Axon i to Neuron k (0 – no connection, 1 – there is connection)

b_k is the leaky potiential of Neuron k

I_k is the Membrane Potential of Neuron k

In this formula, a_{ki} is fixed value:

$$a_{ki} = \begin{cases} -1 & if\ i\ is\ even \\ 1 & if\ i\ is\ odd \end{cases} \quad (5.11)$$

It can be seen that formula (5.10) is almost the same as formula (5.1). Where, a_{ki} and x_i are known values, the problem here is finding c_{ki} and b_k. In other word, instead of training the weights like traditional ANN, Tea will train the connections between

Neurons. Similar to ANN, perform derivatives over c_{ki} and b_k on the Loss function (which should be categorial crossentrophy) to update the new values via gradient descent.

However, c_{ki} represents the connection of Axon i to Neuron k, so this value can only be 0 (no connection) or 1 (connection), which solves by gradient descent this is not possible. Therefore, the rounding rule must be used for the calculation. As follows:

Step 1: Initialize random real numbers c_{ki}^{sh} from 0 to 1

Step 2: Rounding as following:

$$c_{ki} = \begin{cases} 1 & if\ c_{ki}^{sh} > 0.5 \\ 0 & otherwise \end{cases} \quad (5.12)$$

Step 3: Continue to train, update the new value according to the same rounding rule as above.

5.2.3 Outcome

The final result will be a network consisting of many Neurons, in which these Neurons are connected to each other through a rule found by the training process. Neurons only communicate with each other through 0s or 1s instead of float numbers like traditional ANNs. In other words, the information is encoded in the time dimension (Figure 5.6).

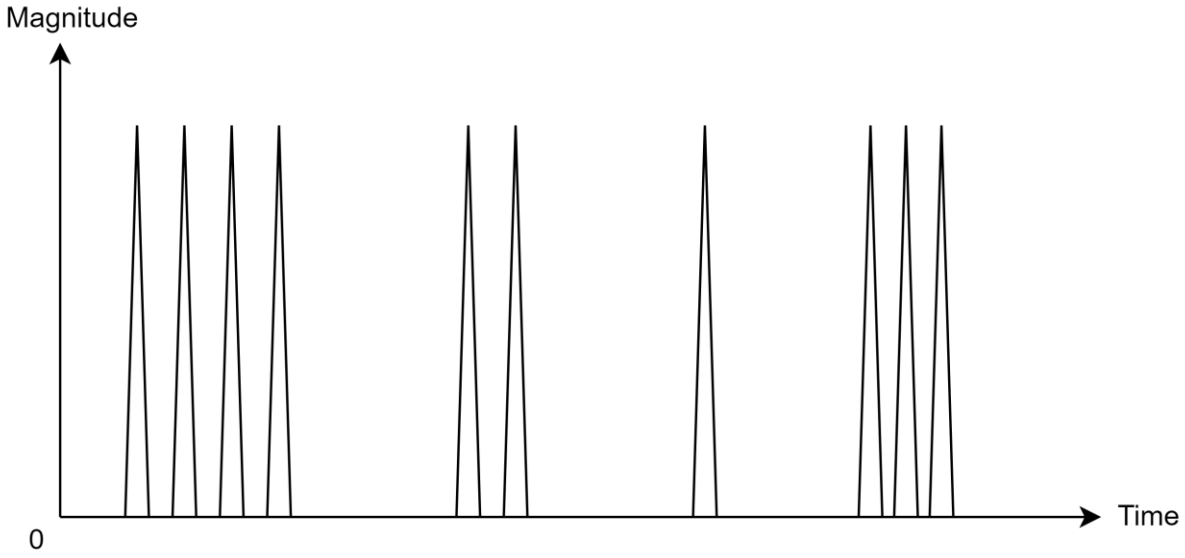


Figure 5.6 Information is encoded in time dimension

5.3 Conclusion

CHAPTER 5 gave an overview of a method by which the SNN will be trained. By using this method, we can apply it to a simple recognition problem, which will be covered in CHAPTER 6.

CHAPTER 6. ASIC DESIGN METHODOLOGY

As mentioned in section 1.2, although SNN has a lot of promise, one of the biggest obstacles is the hardware platform. With today's commercialized chips, it is difficult to implement SNN. The purpose of redesigning architecture in CHAPTER 4 is to deploy the RANC architecture and the SNN Fully Parallel architecture mentioned in CHAPTER 2 and CHAPTER 3, respectively to the ASIC platform, which can bring us closer to creating a completely new SoC system, forming a hardware platform that can perfectly implement SNN with the highest performance and the lowest price. This chapter will cover the methodology that we use to carry out our design on ASIC.

6.1 The idea of designing a chip

Our idea is to design an ASIC with multiple Cores, which are arranged in a matrix form, as shown in Figure 6.1(b). Our chip's computing architecture is not designed based on Von Neuman, whose Central Processing Unit and Memory respectively execute all the calculations and store information (Figure 6.1(a)). In our architecture, each core can process computations and store data independently because it comprises the Router block, the Scheduler block, and the Neuron Grid block, which were mentioned in 2.3.4, 2.3.5, and 4.2. Additionally, all the Cores can have interactions since the Router in each core is responsible for communicating with its nearby Core in the North, South, East, and West.

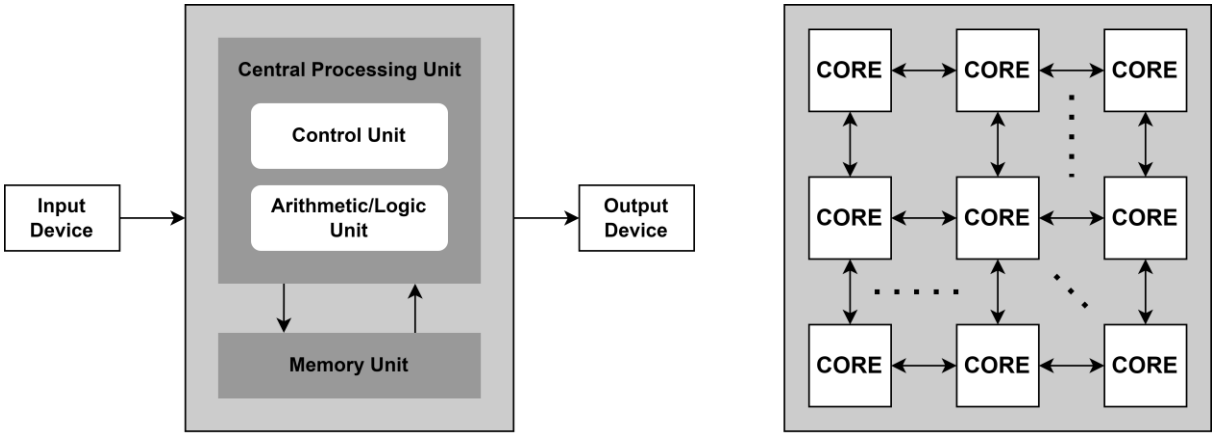


Figure 6.1 (a) Von Neumann architecture (b) Our architecture

Router of Core 0 receives input packets of the chip and transfers them to appropriate Cores based on *dx*, *dy* components in each packet. After transmitting all the packets, Router in each Core passes Axon destination and deliver tick offset to Scheduler and Neuron Grid starts to operate with the *tick*'s command.

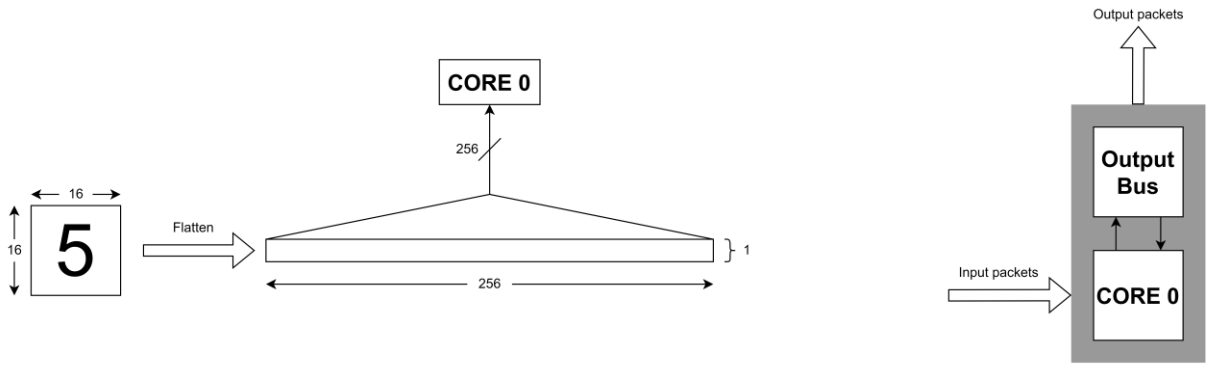
In order to ensure that the chip is practical, our design is verified and implemented by an RTL to GDSII tool, which is mentioned in 6.4.

6.2 Network structure and encoding – decoding method

6.2.1 Network structure

Our aim is to design a non–Von Neuman computation architecture that consists of several Cores connected to each other in the form of a matrix. The matrix form allows this architecture to be applicable and reusable for many different problems. In this section, we use the test set in the MNIST dataset with the size of 28×28 each image to build 2×1, 2×2, 2×3 and 3×3 architectures.

In 2×1 architecture, there is only 1 Core with 1 Output Bus, as shown in Figure 6.2(b). Each Axon among 256 Axons in this Core is processed with a spike which is encoded from the corresponding pixel of a MNIST picture. Hence, we adjusted the resolution of the picture from 28×28 to 16×16 to map with the architecture (Figure 6.2(a)). Receiving spikes, including information of a picture, the Core begins to operate and generates output packets which are transmitted to an output filtering block - Output Bus.

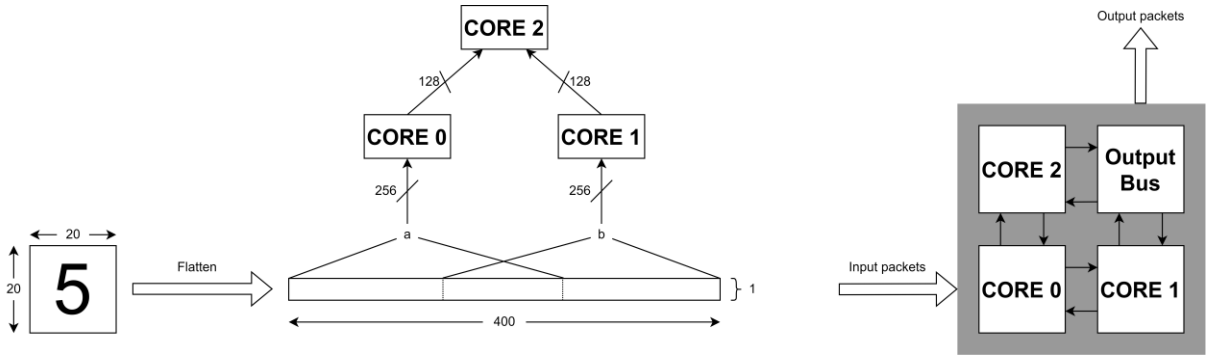


(a) (b)

Figure 6.2 (a) 2×1 netword grid input method (b) 2×1 netword grid structure

In 2×2 architecture (Figure 6.3(b)), there are 3 computation Cores with 1 Output Bus. Similar to 2×1 architecture, each Axon among 256 Axons in each Core is processed

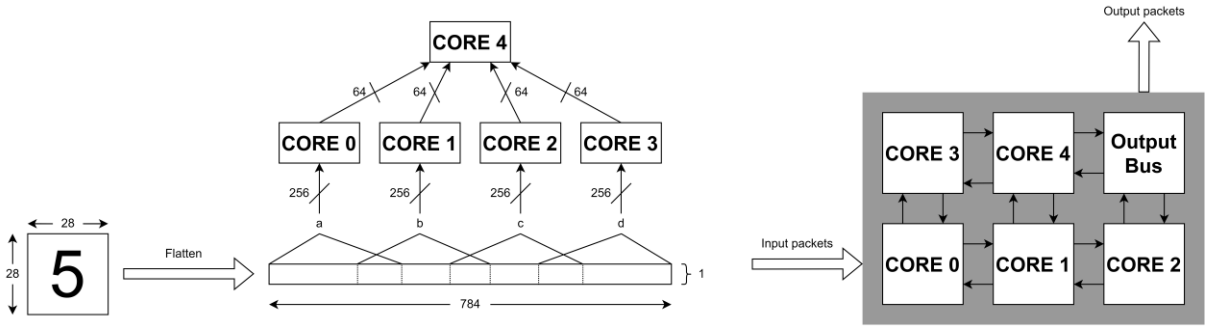
with a spike which is encoded from the corresponding pixel of a MNIST picture. The resolution of pictures used in this architecture is downscale to 20×20 in order that input packets from the pictures are sent to the first two Cores, which belong to the first layer in the network (Figure 6.3(a)). The third Core, which receives the generated spike from the first 128 Neurons of each Core in the first layer, is set in the second layer. Outputs of the Core in layer 2 are transmitted to Output Bus, which releases output packets.



(a) (b)

Figure 6.3 (a) 2×2 network grid input method (b) 2×2 network grid structure

2×3 architecture (Figure 6.4) is designed in same way as what was implemented on 2×2 architecture but with 5 Cores and 1 Output Bus.



(a) (b)

Figure 6.4 (a) 2×3 netword grid input method (b) 2×3 netword grid structure

3×3 architecture (Figure 6.5) is similar to 2×3 except for some null Cores which are not used.

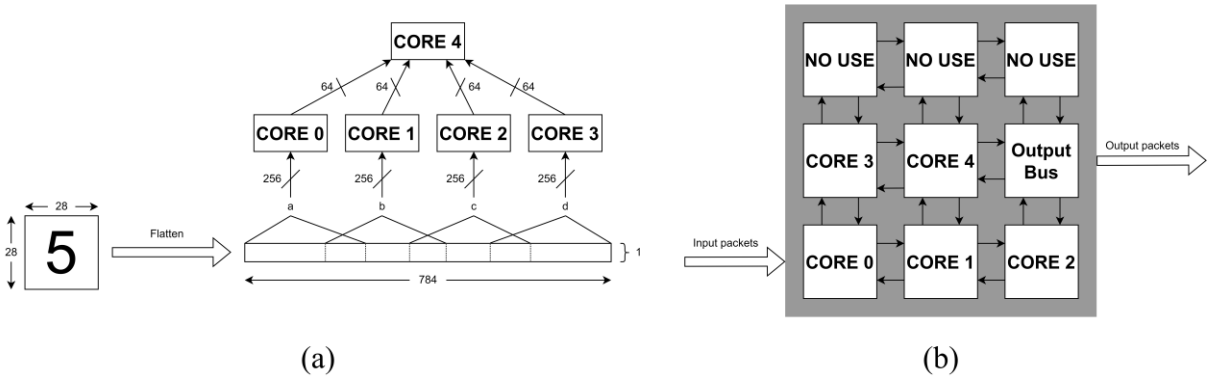


Figure 6.5 (a) 3×3 network grid input method (b) 3×3 network grid structure

6.2.2 Input encoding method

We apply this architecture for a classic function - handwritten image recognition. Inputs of the network are images from MNIST dataset, whose resolution is 28×28 pixels. Each image illustrates a random handwritten number from 0 to 9. Therefore, our hardware architecture simulates a SNN as a fully connected layer in CNN. Because of the simple task, we do not extract the feature but apply the images directly to the architecture with the following encoding method:

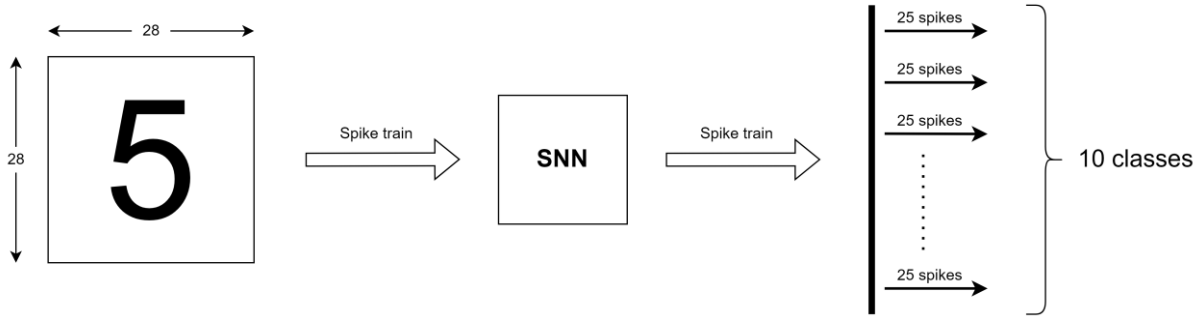


Figure 6.6 SNN recognition flow

- All the pixels with their value greater than or equal to 126 are normalized to 1.
- All pixels with their value less than 126 are normalized to 0.
- All pixels with normalized value 0 can be altered.
- All pixels with normalized value 1 are encoded into 30-bit packets, which are sent into SNN after *tick* is enabled.
- Contents of the packets depending on the number of input Cores and image distribution based on those Cores.

6.2.3 Output decoding method

The last computation Core in the network only uses 250 Neurons instead of 256 Neurons to process with the aim that every 25 Neurons vote for a number from 0 to 9 (Figure 6.6). The Neuron in position n raises a vote that the input picture demonstrates number $n \ mod \ 10$. For example, 2^{nd}, 12^{nd}, 22^{nd}, 32^{nd}, ... Neurons are used to vote for number 2. The number which has the highest votes is chosen as the prediction of the network at the current *tick*. Specifically, the vote mentioned in this method is an illustration of spikes. A Neuron generating a spike means that it raises a vote. In contrast, no spike released from a Neuron shows that it abstains from voting.

6.3 Architectural verification

The architecture is simulated with 10,000 handwritten digit images in the MNIST dataset by Modelsim.

From the simulation process, we have the following table result:

Table 6.1 Throughput and Accuracy result

Architecture	Num of clks	Accuracy	Architecture	Num of clks	Accuracy
SNN Parallel 2×1	730	91.38%	SNN Sequential 2×1	66050	91.38%
SNN Parallel 2×2	890	94.01%	SNN Sequential 2×2	66050	94.01%
SNN Parallel 2×3	1004	95.18%	SNN Sequential 2×3	66050	95.18%
SNN Parallel 3×3	1004	95.18%	SNN Sequential 3×3	66050	95.18%

In essence, 2×3 grid and 3×3 grid are the same, only different in size like a phone but wearing 2 different cases. Therefore, their results are the same.

The number of clock cycles to compute and give an output is 730 for 2×1 Parallel architecture, 890 for 2×2 Parallel architecture, and 1004 for 2×3 (3×3) Parallel architecture, respectively, while they are 66050 for all Sequential architectures. It can be seen that the simulation results completely match the theoretical calculation. For the SNN Sequential architecture, the number of clock cycles required for detecting one picture through the simulation is identical to the calculation in section 2.3.2. However, the number of clock cycles for the detecting process of SNN Parallel architecture through the simulation is higher than the calculation in section 4.2.4.2 a), which is evident due to the congestion in the network.

Compared with the labels of the MNIST images, the results are obtained. The accuracy of 2×1, 2×2, 2×3 (3×3) architecture is respectively 91.38%, 94.01%, and 95.18% - which are impressive results with such a simple architecture. This shows the huge potential of this architecture when integrating a larger number of Cores.

6.4 Asic implementation

6.4.1 Tools overview

a) Sky130 PDK

A process design kit (PDK) is a set of files used within the semiconductor industry to model a fabrication process for the design tools used to design an integrated circuit. The PDK is created by the foundry defining a certain technology variation for their processes. It is then passed to their customers to use in the design process. The customers may enhance the PDK, tailoring it to their specific design styles and markets. The designers use the PDK to design, simulate, draw and verify the design before handing the design back to the foundry to produce chips. The data in the PDK is specific to the foundry's process variation and is chosen early in the design process, influenced by the market requirements for the chip. An accurate PDK will increase the chances of first-pass successful silicon [46].

The SkyWater Open Source PDK is a collaboration between Google and SkyWater Technology Foundry to provide a fully open-source Process Design Kit and related resources, which can be used to create manufacturable designs at SkyWater’s facility [47].

Figure 6.7 Skyware 130nm PDK

b) OpenLane

OpenLane is an automated register-transfer level (RTL) to GDSII flow based on several components including OpenROAD, Yosys, Magic, Netgen, CVC, SPEF-Extractor, CU-GR, Klayout and several custom scripts for design exploration and optimization. The flow performs full ASIC implementation steps from RTL all the way down to GDSII [48].

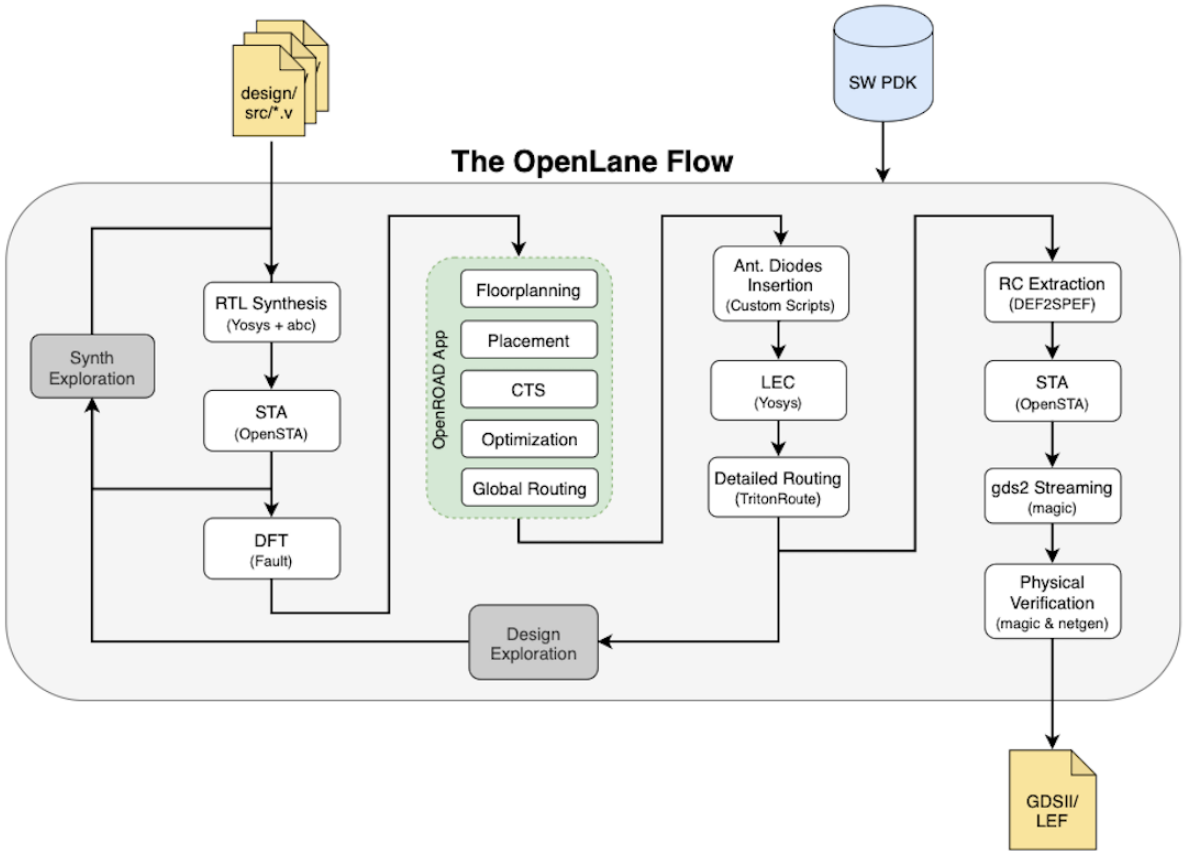


Figure 6.8 The OpenLane Flow

Figure 6.8 shows the full flow of OpenLane.

6.4.2 Result

6.4.2.1 General result

After optimizing with all the changes in architecture mentioned in the above sections, the network has been comprehensively simulated and synthesized, ensuring that the functions and computation results remain the same as the original architecture.

Functional testing and comparison are performed on an MNIST set of 10,000 handwritten alphanumeric images with the same set of training parameters. To compare

the parameters of processing speed, circuit area, and power consumption, the architectures before and after making various changes in the network are synthesized by the following tools and specifications:

- Integrated software: Openlane.
- Technology: 130nm SkyWater SKY130 PDK.
- Library: SKY130 High Density Digital Standard Cells.
- Network size: Grid network of 2×1, 2×2, 2×3 and 3×3 Cores.

From the calculation results in Table 6.1 and the above specifications, the ASIC-making process can be jogged on correctly. Accordingly, the final synthesis result is shown in Table 6.2. The table lists the ASIC specification of the SNN Parallel architecture and the SNN Sequential one, with four kinds of networks for each model.

Table 6.2 ASIC synthesis result

	Throughput					
	25 pic/s			**60 pic/s**		
Architecture	**Area (μm^2)**	**Power (W)**	**Frequency (Hz)**	**Area (μm^2)**	**Power (W)**	**Frequency (Hz)**
SNN Parallel 2×1	2,459,883	2.78×10^{-5}	18,250	2,460,084	6.61×10^{-5}	43,800
SNN Sequential 2×1	1,584,948	1.34×10^{-3}	1,651,250	1,584,656	3.21×10^{-3}	3,963,000
SNN Parallel 2×2	6,605,602	9.72×10^{-5}	22,250	6,604,600	2.32×10^{-4}	53,400
SNN Sequential 2×2	4,634,659	4.29×10^{-3}	1,651,250	4,635,561	1.03×10^{-2}	3,963,000
SNN Parallel 2×3	8,965,143	1.40×10^{-4}	25,100	8,964,653	3.35×10^{-4}	60,240
SNN Sequential 2×3	7,287,158	6.68×10^{-3}	1,651,250	7,286,926	1.60×10^{-2}	3,963,000
SNN Parallel 3×3	13,023,418	2.12×10^{-4}	25,100	13,023,557	5.05×10^{-4}	60,240
SNN Sequential 3×3	10,505,573	9.82×10^{-3}	1,651,250	10,511,991	2.48×10^{-2}	3,963,000

Specifically, within the required image recognition speed of 25 pictures/second, the area needed for a 2×1 SNN Parallel network is about 55% more than a 2×1 SNN Sequential network. The number becomes 43% for 2×2 networks and 23% for 2×3 networks. When the size of networks significantly increases, the difference gradually narrows. Notably, for a 3×3 network, the SNN Parallel model only takes about 24% more area than a SNN Sequential model. The same observation can be seen when the synthesis specifications are modified to obtain the throughput of 60 pictures/second.

In contrast to area utilization, in terms of power consumption, to process the same throughput, the amount of power used by a SNN Parallel network is much lower than by a SNN Sequential network. Precisely, by the size of a 2×1 network, to process a throughput of 25 pictures/second, a SNN Parallel network only uses about 2% of the

total power consumed by the same size SNN Sequential network. For larger network sizes, the number is also just over 2%. A similar result is also observed for the throughput of 60 pictures/second, i.e., just over 2%.

For the last criteria, the processing frequency of each architecture, a SNN Parallel design also outperforms the same size SNN Sequential network. For a more good-looking comparison, the ratio of a SNN Parallel network over the same size SNN Sequential network is generally only around 1.3%. In detail, with the network size of 2×1, 2×2, 2×3, and 3×3, the number is 1.10%, 1.35%, 1.52%, and 1.52%, respectively, for both two levels of throughput.

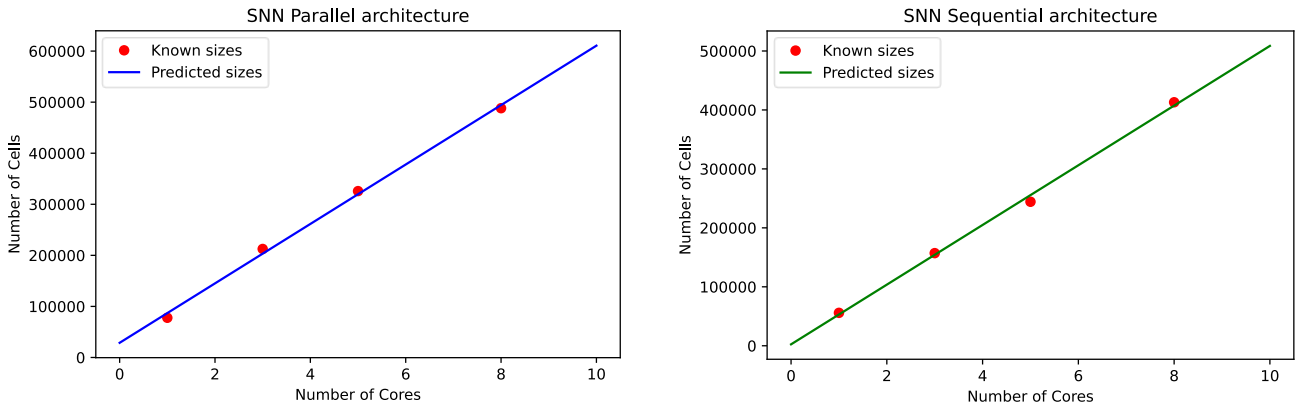
6.4.2.2 Hardware resources

a) Number of cells

Table 6.3 Number of cells for each architecture

Network size	Number of cells	Network size	Number of cells
SNN Parallel 2×1	77,716	SNN Sequential 2×1	55,834
SNN Parallel 2×2	212,509	SNN Sequential 2×2	157,119
SNN Parallel 2×3	325,803	SNN Sequential 2×3	244,245
SNN Parallel 3×3	488,125	SNN Sequential 3×3	413,019

Table 6.3 shows the results for the number of cells used to make up the architectures. Based on the data in this table, we were able to draw two graphs that are shown in Figure 6.9, depicting the correlation between the size of each architecture and the number of cells to make up those architectures.



(a) (b)

Figure 6.9 The relationship between architecture size and number of cells

As can be seen, both graphs depict a linear function with dots representing known dimensions (data from Table 6.3). Using the Linear Regression algorithm from machine learning, we found the two best-fit functions (2 lines in Figure 6.9). From here, we can easily predict the number of cells to form an architecture with a size of $N \times N$.

SNN Parallel architecture's number of cells equation:

$$n_{cell} = n_{Core} \times 58190.94 + 28726.74 \quad (6.1)$$

SNN Sequential architecture's number of cells equation:

$$n_{cell} = n_{Core} \times 50622.27 + 2409.6 \quad (6.2)$$

Where:

n_{core} is the number of Core in the architecture. Since each architecture include an Output Bus, so: $n_{Core} = architecture's\ size - 1$. For example, a 3×3 grid has $3 \times 3 - 1 = 8$ Cores.

n_{cell} is the number of cells to make up the architecture

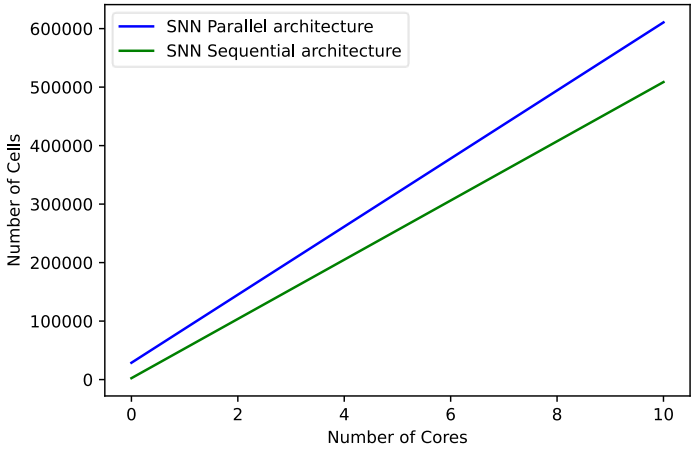


Figure 6.10 Correlation of the number of cells between parallel architecture and sequential architecture

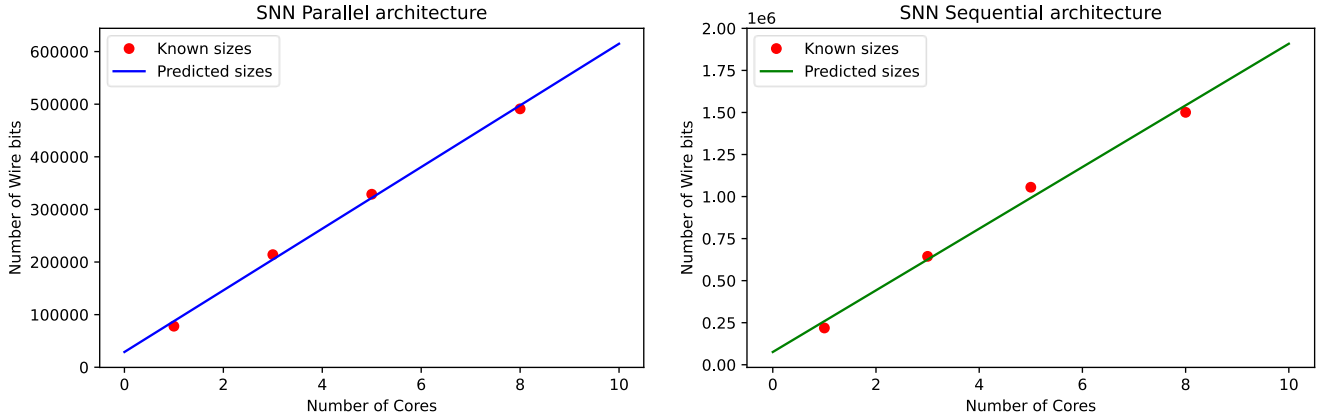
From the graph shown in Figure 6.10, as the architecture size increases, the difference in the number of cells between SNN Parallel and SNN Sequential becomes larger. However, this difference does not change too quickly (with the number of Cores from 0 to 10, 2 lines are almost parallel).

b) Number of wire bits

Table 6.4 Number of wire bits for each architecture

Network size	Number of wire bits	Network size	Number of wire bits
SNN Parallel 2×1	78,014	SNN Sequential 2×1	218,678
SNN Parallel 2×2	214,022	SNN Sequential 2×2	644,466
SNN Parallel 2×3	328,838	SNN Sequential 2×3	1,055,387
SNN Parallel 3×3	491,133	SNN Sequential 3×3	1,500,636

Similarly, Table 6.4 shows the number of wire bits in each architecture. From the data in this table, we can again draw the following 2 graphs to predict the number of wires for each size.



(a) (b)

Figure 6.11 The relationship between architecture size and number of wire bits

The correlation between the number of wires and the number of cores of each architecture is also generalized into two equations as follows:

SNN Parallel architecture's number of wire bits equation:

$$n_{wb} = n_{Core} \times 58590.81 + 28990.79 \quad (6.3)$$

SNN Sequential architecture's number of wire bits equation:

$$n_{wb} = n_{Core} \times 183276.23 + 75867.76 \quad (6.4)$$

Where:

n_{wb} is the number of wire bits inside the architecture

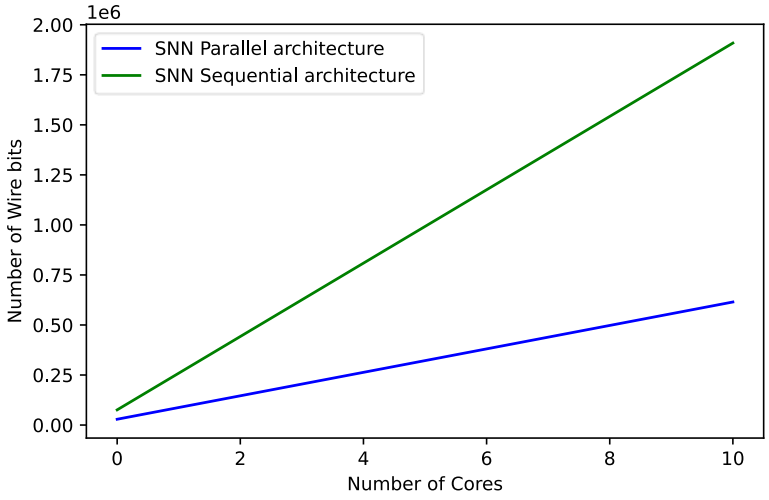


Figure 6.12 Correlation of the number of wire bits between parallel architecture and sequential architecture

Figure 6.12 shows the comparison between the number of wire bits and the number of Cores of each architecture. The difference in the number of wires changes very quickly when the size of the two architectures changes. When the architecture is larger, the number of wires that SNN Sequential requires is much higher than SNN Parallel.

6.5 Conclusion

CHAPTER 6 detailed the steps to synthesize architecture on the ASIC platform:

Initially, from the idea of building a non-von Neumann general architecture system, simulated network models were built to conduct MNIST image set recognition.

These network models are then simulated and tested for functionality and accuracy.

Finally, these models are synthesized on the ASIC platform using open-source tools based on two architectural platforms that are RANC in CHAPTER 2 and SNN Fully Parallel in CHAPTER 3. The results on energy, frequency, area, number of cells, and number of wires are compared and evaluated visually.

The Parallel architecture has brought a much better performance than Sequential; the price to pay is the increase in circuit area along with the number of cells.

CONCLUSION AND FUTURE WORK

In this project, I have completed the initial requirements, specifically as follows:

- In CHAPTER 1: I had an overview of the theories that I needed to know to be able to carry out the project.
- In CHAPTER 2: I learned in detail about RANC architecture.
- In CHAPTER 3: I learned in detail about SNN Fully Parallel architecture.
- In CHAPTER 4: I optimized the RANC architecture and the SNN Fully Parallel architecture, rearranged the modules to be more suitable for ASIC synthesis.
- In CHAPTER 5: I learned a way to train an SNN model.
- In CHAPTER 6: I came up with the idea to create a chip based on the redesigned architecture and built networks to simulate and successfully synthesize those architectures on open-source software.

In short, I have accomplished my original goals, which are “redesign, optimize, test, and synthesize them on the ASIC platform to make something new”.

Despite the cost in terms of area, the power utilization and processing frequency of the Fully Parallel architecture are superior to the original RANC's proposed design.

However, in the architecture that I redesigned, the data of the Neurons is not reloadable; they are fixed in the flipflops (when reset, they are automatically initialized to the given value). In order to be able to actually make the whole SoC system, in the future, this architecture needs to add a method to be able to reload the Neurons’s data.

REFERENCES

[1] H. Beck, "The brain works with 20 watts. This is enough to cover our entire thinking ability.," [Online]. Available: https://www.munichre.com/topics-online/en/digitalisation/interview-henning-beck.html.

[2] Wikipedia, The Free Encyclopedia, "Computational neuroscience," 17 June 2022. [Online]. Available: https://en.wikipedia.org/wiki/Computational_neuroscience.

[3] T. P. Trappenberg, in *Fundamentals of computational neuroscience*, United States, Oxford University Press Inc, 2010, p. 2.

[4] W. Gerstner, W. Kistler, R. Naud and L. Paninski, Neuronal Dynamics, Cambridge, UK: Cambridge University Press, 2014.

[5] P. Dayan and L. F. Abbott, Theoretical neuroscience: computational and mathematical modeling of neural systems, Cambridge: MIT Press, 2001.

[6] P. S. Churchland, C. Koch and T. J. Sejnowski, "What is computational neuroscience?," in *Computational Neuroscience*, MIT Press, 1993, p. 46–55.

[7] B. Gutkin, D. Pinto and B. Ermentrout, "Mathematical neuroscience: from neurons to circuits to systems," *Journal of Physiology-Paris. Neurogeometry and visual perception,* p. 209–219, 2003-03-01.

[8] N. Kriegeskorte and P. K. Douglas, "Cognitive computational neuroscience," *Nature Neuroscience,* September 2018.

[9] M. PAPADATOU-PASTOU, "Encephalos Journal," *www.encephalos.gr.*

[10] E. D. Paolo, "Organismically-inspired robotics: homeostatic adaptation and teleology beyond the closed sensorimotor loop," in *Semantic Scholar*.

[11] R. Brooks, D. Hassabis, D. Bray and A. Shashua, "Turing centenary: Is the brain a good model for machine intelligence?," *Nature,* vol. 482, no. 7386, p. 462–463, 2012-02-22.

[12] A. Browne, Neural Network Perspectives on Cognition and Adaptive Robotics, CRC Press, 1997-01-01.

[13] A. Gilra, "Biologically-plausible learning in neural networks for movement control and cognitive tasks," in *Institute of Science and Technology Austria*, Mondi Seminar Room 1, Central Building, 2019.

[14] M. Zorzi, A. Testolin and I. P. Stoianov, "Modeling language and cognition with deep unsupervised learning: a tutorial overview," *Frontiers in Psychology,* 2013-08-20.

[15] A. Shai and M. E. Larkum, "Branching into brains," *eLife,* 2017-12-05.

[16] Wikipedia, The Free Encyclopedia, "Neuromorphic engineering," 26 June 2022. [Online]. Available: https://en.wikipedia.org/wiki/Neuromorphic_engineering.

[17] D. Monroe, "Neuromorphic computing gets ready for the (really) big time," *Communications of the ACM,* 2014.

[18] Zhao, W. S.; Agnus, G.; Derycke, V.; Filoramo, A.; Bourgoin, J. -P.; Gamrat, C., " Nanotube devices based crossbar architecture: Toward neuromorphic computing," *Nanotechnology,* 2010.

[19] HumanBrainProject, "The Human Brain Project SP 9: Neuromorphic Computing Platform," Youtube, 2013. [Online]. Available: https://www.youtube.com/watch?v=6RoiZ90mGfw.

[20] Mead Carver, "Neuromorphic electronic systems," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1629-1636, Oct. 1990.

[21] R. A. Alzahrani and A. C. Parker, "Neuromorphic Circuits With Neural Modulation Enhancing the Information Content of Neural Signaling," in *ICONS 2020*, Online, 28 July 2020.

[22] A. K. Maan, D. A. Jayadevi and A. P. James, "A Survey of Memristive Threshold Logic Circuits," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 28, no. 8, pp. 1734-1746, Aug. 2017.

[23] Y. Zhou and S. Ramanathan, "Mott Memory and Neuromorphic Devices," *Proceedings of the IEEE,* vol. 103, no. 8, pp. 1289-1310, Aug. 2015.

[24] Eshraghian, Jason K.; Ward, Max; Neftci, Emre; Wang, Xinxin; Lenz, Gregor; Dwivedi, Girish; Bennamoun, Mohammed; Jeong, Doo Seok; Lu, Wei D., "Training Spiking Neural Networks Using Lessons from Deep Learning," October 1, 2021.

[25] Hananel-Hazan, "bindsnet: Simulation of spiking neural networks (SNNs) using PyTorch," 31 March 2020. [Online]. Available: https://github.com/BindsNET/bindsnet.

[26] S. K. Boddhu and J. C. Gallagher, " Qualitative Functional Decomposition Analysis of Evolved Neuromorphic Flight Controllers," *Applied Computational Intelligence and Soft Computing,* p. 1–21, 2012.

[27] C. Mead, "carvermead," [Online]. Available: http://www.carvermead.caltech.edu/index.html.

[28] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks,* p. 1659–1671, 1997.

[29] W. Gerstner, in *Spiking neuron models : single neurons, populations, plasticity*, Cambridge, U.K., Cambridge University Press, 2002, p. 1969.

[30] V. Lyashenko, "Basic Guide to Spiking Neural Networks for Deep Learning," [Online]. Available: https://cnvrg.io/spiking-neural-networks/.

[31] Yann LeCun, Courant Institute, NYU Corinna Cortes, Google Labs, New York Christopher J.C. Burges, Microsoft Research, Redmond., "THE MNIST DATABASE of handwritten digits," [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[32] "Support vector machines speed pattern recognition," *Vision Systems Design.*

[33] S. Gangaputra, "Handwritten digit database," [Online]. Available: https://www.cis.jhu.edu/~sachin/digit/digit.html.

[34] Y. Qiao, "THE MNIST DATABASE of handwritten digits," 2007. [Online].

[35] J. C. Platt, "Using analytic QP and sparseness to speed training of support vector machines," *Advances in Neural Information Processing Systems,* p. 557–563, 1999.

[36] P. J. Grother, "NIST Special Database 19 - Handprinted Forms and Characters Database," *National Institute of Standards and Technology.*

[37] E. Kussul and T. Baidyk, "Improved method of handwritten digit recognition tested on MNIST database," *Image and Vision Computing,* p. 971–981, 2004.

[38] B. Zhang and S. N. Srihari, "Fast k-Nearest Neighbor Classification Using Cluster-Based Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* p. 525–528, 2004.

[39] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[40] electronics-notes.com, "What is an ASIC: application specific integrated ciruit," [Online]. Available: https://www.electronics-notes.com/articles/electronic_components/programmable-logic/what-is-an-asic-application-specific-integrated-circuit.php.

[41] Filipp Akopyan et al., "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip,* vol. 34, no. 10, pp. 1537-1557, Oct. 2015.

[42] J. Mack et al., "RANC: Reconfigurable Architecture for Neuromorphic Computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 40, no. 11, pp. 2265-2278, Nov. 2021.

[43] V. Truong-Tuan et al., "FPGA Implementation of Parallel Neurosynaptic Cores for Neuromorphic Architectures," *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS),* pp. 1-4, 2021.

[44] V. Salauyou, "Using ASMD-FSMD Technique for Digital Device Design," in *Theory and Engineering of Dependable Computer Systems and Networks*, May 2021, pp. 391-401.

[45] Svitla Team, "Modern methods of neural network training," 14 January 2020. [Online]. Available: https://svitla.com/blog/modern-methods-of-neural-network-training.

[46] Wikipedia, The Free Encyclopedia, "Process design kit," 8 June 2022. [Online]. Available: https://en.wikipedia.org/wiki/Process_design_kit.

[47] Google, "SkyWater Open Source PDK," Google, [Online]. Available: https://github.com/google/skywater-pdk.

[48] Google, "OpenLane," Google, [Online]. Available: https://github.com/The-OpenROAD-Project/OpenLane.