**Circuits and Systems**
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

**CAS-2022-5346371**

# M.Sc. Thesis

# Off-chip Self-timed SNN Custom Digital Interconnect System

**Yichen Yang B.Sc.**

**Abstract**

To support the spike propagates between neurons, neuromorphic computing systems always require a high-speed communication link. Meanwhile, spiking neural networks are event-driven so that the communication links normally exclude the clock signal and related blocks. This thesis aims to develop a self-timed off-chip interconnect system with ring topology that supports multi-point communication in neuromorphic computing systems. This interconnect system is implemented in high-level modeling with SystemC and involves the burst-mode two-wire protocol in point-to-point communication. In order to ensure the flexibility of the system, the distributed control system is involved. Further, the system can be configured with different numbers of chiplet to fulfill various spiking neural network structures. We also explore optimization methods, which is a bi-directional ring topology achieving the growth of throughput. Based on evaluation and simulation results, the interconnect system can achieve 4.302Gbps with the specific application scenario.

# Off-chip Self-timed SNN Custom Digital Interconnect System

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Yichen Yang B.Sc.
born in Xi'an, China

This work was performed in:

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

**Delft University of Technology**

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Off-chip Self-timed SNN Custom Digital Interconnect System"** by **Yichen Yang B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 28.11.2022

Chairman: _____
Prof. Dr. Ir. Rene van Leuken

Advisor: _____
Dr. Aditya Dalakoti

Committee Members: _____
Dr. Charlotte Frenkel

_____

# Abstract

To support the spike propagates between neurons, neuromorphic computing systems always require a high-speed communication link. Meanwhile, spiking neural networks are event-driven so that the communication links normally exclude the clock signal and related blocks. This thesis aims to develop a self-timed off-chip interconnect system with ring topology that supports multi-point communication in neuromorphic computing systems. This interconnect system is implemented in high-level modeling with SystemC and involves the burst-mode two-wire protocol in point-to-point communication. In order to ensure the flexibility of the system, the distributed control system is involved. Further, the system can be configured with different numbers of chiplet to fulfill various spiking neural network structures. We also explore optimization methods, which is a bi-directional ring topology achieving the growth of throughput. Based on evaluation and simulation results, the interconnect system can achieve 4.302Gbps with the specific application scenario.

# Acknowledgments

This thesis marked the end of my student's life at TU Delft. During the development of the whole project, I have receievd a great deal of support and assistance from many people.

I would like to express my deepest appreciation to my supervisor Prof. Dr. Ir. Rene van Leuken, who provided invaluable patience and feedback on my research. Also, I could not have undertaken this journey without the advisors from Innatera Nanosystems. Special thanks to Dr. Aditya Dalakoti, Dr. Kamlesh Kumar Singh, Ir. Alexander de Graaf, and Jinbo Zhou. Without their assistance and dedicated involvement in every step throughout the process, this project would have never been accomplished. Additionally, they bring me many knowledge and practical skills in the field of digital design

I am also grateful to my parents. Thanks to their unfailing support and continuous encouragement, I am able to complete my studies in the Netherlands. Also, special thanks to my colleagues and friends, Tianyu, Jiongyu, etc. Working with them has brought a lot of joy into my life. Last but not least, I would like to thank my girlfriend, Jin. In these months, she is always there for me.

Yichen Yang B.Sc.
Delft, The Netherlands
28.11.2022

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Problem Statement

Nowadays, neuromorphic computing system has become a new noticeable formal computing system, which mimics the physics of the human brain and nervous system with the help of the Spiking Neural Network (SNN). The neural array is a prominent part of that system, and tremendous spikes signals are transferred between those neural arrays to simulate the connection system in the brain. Therefore, it would be essential to conduct a particular interconnect system for that system to ensure the communication demands are satisfied.

This thesis aims to develop a high-speed, self-timed interconnect system between chiplets in a neuromorphic system. Normally, neural arrays, which are packaged in chiplets, generate the spikes signals with high width, and they are needed to be transferred in an extremely short time, which means that it is necessary to keep the system working at high speed. Meanwhile, there are considerable benefits when a neuromorphic system is based on the self-timed instead of relying on the clock. For example, the speed of interconnect is only limited by wire speed instead of the clock frequency. Also, self-timed has some positive effect on power consumption. Consequently, high speed and self-timed will be two main characteristics of interconnect systems.

Address Event Representation (AER) is a widely used protocol in neuromorphic interconnect systems, which focuses on point-to-point communication and propagates the data based on event address. In the AER circuit, as an event or spike only has one destination, the communication between two neural arrays can be seen as the Point to Point communication. Meanwhile, there will be an event if a one-bit signal generates a spike, and then that event is encoded to the address information used to transmit to the receiver. After that, the receiver-end decodes the address information to represent the original event. So, these characteristics lead to some bottlenecks when applying AER to a neuromorphic system that contains many neural arrays and more complex communication demands.

Compared with AER, our interconnect system not only supports Point to Point communication between any two chiplets but also introduces some mechanism to support multi-point communication between several chiplets. Also, a more effective method to represent the event is conducted to replace the AER. In our systems, a multi-points communication mechanism is designed to ensure the demand for collaboration between chiplets, as there are many neural arrays in a neuromorphic system generally, which means there is more than one chiplet.

In addition, a bi-directional ring topology and the local controller are introduced to ensure the interconnect system works with low latency and keeps synchronizing between transmitters and receivers. Every neural array, as known as chiplet, contains

a transmitter and a receiver to finish spikes signal transmission with a feed-forward protocol. Meanwhile, they are connected with a bus that has a ring topology so that it can guarantee the spikes signal could arrive at any receiver and also keeps the low latency of the whole system. As for the local controller, a state machine is introduced. It would work with a round-robin arbiter to determine the permission for data transmission, and the feed-forward design would keep synchronization between transmitters and receivers.

## 1.2 Goal

This thesis project aims to design and implement a customized high-speed burst-mode asynchronized interconnect system for a neuromorphic computing system and propose the optimization method to enhance the system's performance. Additionally, evaluate the power, area, and throughput of interconnect system and verify the functions with multiple MNIST use cases.

## 1.3 Contributions

The contributions of this thesis include the following parts:

- Implement the basic SerDes Link for two neural arrays.
- Based on the designed SerDes link, propose the ring topology for the off-chip interconnect system between multiple chiplets in a neuromorphic system and implement it in system modelling level.
- Explore and implement the potential optimization method for the off-chip interconnect system.
- Build the simulation environment and verify the system with a specific MNIST use case .
- Evaluate the system's throughput, area and power, and also compare the result between different two kinds of implementation.

## 1.4 Thesis Outline

- Chapter 2:

  It introduces the background of interconnect systems for neuromorphic computing systems and some detailed methods for implementation.

- Chapter 3:

  It introduces the whole topology of our interconnect system, illustrates the design methodology of the transmitter, receiver and local controller, and elaborates

on how they cooperate with each other to meet the system's functional requirements. Meanwhile, this chapter introduces the optimization methods and how to configure the system according to the customized requirements.

- Chapter 4:

  It shows the simulation results and evaluates system's power, area and throughput. Also, the comparison between different implementations is carried out in this chapter

- Chapter 5:

  It concludes our interconnect system and proposes potential improvements in the future.

# 2 Background

First, the structure of a neuromorphic computing system, which comprises multiple neurons and event-driven communication with a lot of connectivity, will be introduced in this chapter. Furthermore, the interconnect system with different characteristics and working principles are introduced. Meanwhile, some research related to the topology and routing system of communication systems is introduced.

## 2.1 Neuromorphic computing system

With the continuous improvement of transistor technology, a larger number of transistors can be accommodated in the same area of the chip. At the same time, the power consumption of the system is also rising. Therefore, in order to address this problem, the neuromorphic computing system is proposed, which is inspired by the brain's structure. Meanwhile, Spiking Neural Network (SNN) processing is used in neuromorphic compute accelerator ICs, which use stateful neuron models that share information in the form of sparse asynchronous events (spikes)[16]. State-of-the-art implementations of neuromorphic systems are based on analog, digital, or hybrid mixed-signal silicon technology[17][18].

Generally, the neural system consists of the neuron cell, axons, and synapses, as shown in figure 2.1. The neuron transmits the electricity through the axon and the signal is received by the next neuron through the synapse, which is the junction between several types of neurons. Basically, axons are used by the many types of neurons to form long-distance connections[1] and emulate the interconnectivity[19]. And these axons are represented by the communication link in neuromorphic computing systems. In the following content, we will demonstrate the various neuron models and how these neurons communicates based on the Address-Event Representation (AER) protocol.

### 2.1.1 Neurons model

The neurons require the communication link with specific characteristics. Before introducing the communication link, it is essential to explore the neuron architecture and the working principle of them. There are two most widely used single neuron models in theoretical neuroscience, which are integrate-and-fire model (IF), modeling neurons at an abstract level and the Hodgkin–Huxley (HH) model describing the biophysical mechanisms of cells [20]. In the following content, we will introduce them respectively.

- Integrate-and-fire model

  The integrate-and-fire neuron model was developed by Lapicque and it is one of the most widely used models for analyzing the behavior of neural systems [21].


Figure 2.1: The neural system in brain[1]

The figure 2.2 demonstrates the prototype of this model. This model consists of capacitance C and membrane resistance R, which are shown in figure2.2(A). On the left side, the analogue simulation result of IF model is shown, which reveals the fluctuation of the membrane potential V is related to the input current I.

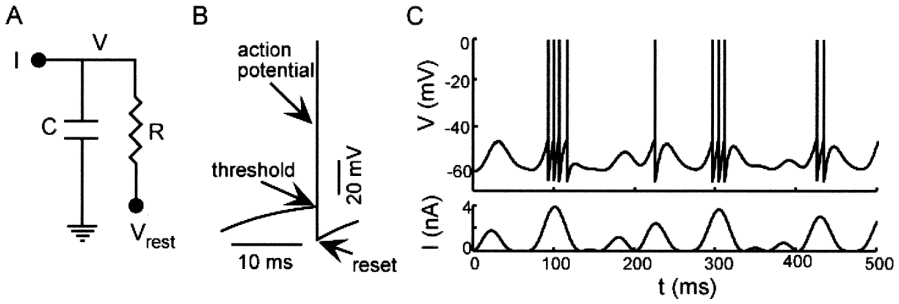


Figure 2.2: The integrate-and-fire model of Lapicque[2]

[21] described the dynamics of the neuron's membrane potential, v(t), which follow the equation 2.1.

$$C_m \frac{dv(t)}{dt} = I_{leak}(t) + I_s(t) + I_{inj}(t) \tag{2.1}$$

$C_m$ is the membrane capacitance. In terms of the current $I$, they represent the synaptic inputs and the injected current. As soon as the membrane potential arrives at a specific threshold, a spike will be generated.

- Hodgkin–Huxley model

  Hodgkin–Huxley model was developed in 1952, which depicts the initiation and spread of action potentials in neurons. Also, it is a mathematical model that aims to explain the ionic mechanisms underlying the initiation and propagation

of action potentials in the squid giant axon[22]. Figure 2.3 can be used to explain the fundamental mechanism of Hodgkin–Huxley model. Every part is regarded as an electrical element. It has three distinct processes: leak channels, voltage-gated ion channels, and pumps and exchangers. Comparatively speaking, this paradigm is more difficult to implement in silicon.

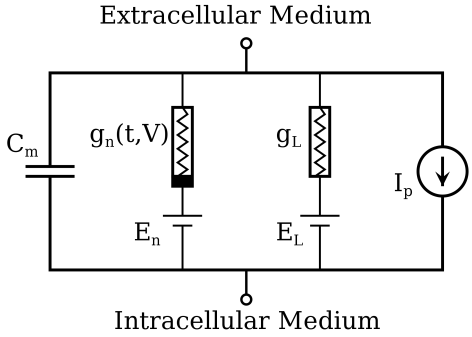


Figure 2.3: The Hodgkin–Huxley model[3]

### 2.1.2 Address-Event Representation

Address-Event Representation(AER) is invented by Mahowald[23], which is a circuit that are used to provide multiplexing/demultiplexing functionality for spikes that are generated by/delivered to an array of individual neurons [1]. Figure 2.4 reveals the architecture of the AER circuit that cooperating with the two neural arrays and transmitting data in serial. Basically, the AER circuit can be divided into three parts, which are multiplexer and encoder placed at the transmitter end, and the demultiplexer and decoder locating at receiver end, and the communication link driven by events.

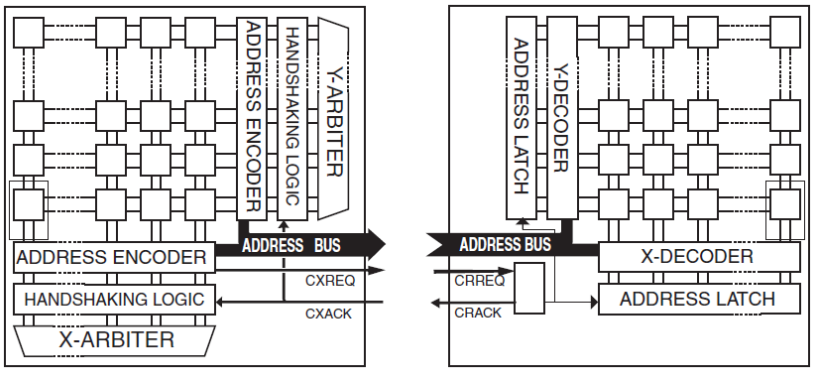


Figure 2.4: Address-Event Representation [1]

The transmitter end aims to encode the spike/data from the neurons, which are able to indicate the address of the fired neuron. In addition, the encoding result is sent out in serial. Also, the proper arbitration mechanisms is essential to avoid the collision of data and there are various arbiter to accomplish this goal in the transmitter end, such as tree arbiter, mesh arbiter, etc.

A scalable multiple-access transmitter is proposed by [4], which supports communication binary activity between two-dimensional arrays. Figure 2.5 reveals the architecture of this transmitter. We can notice that the TX is able to read all active cells in a selected row in parallel. Meanwhile, the burst-mode communication with the three wires protocol is involved in this structure. A row-request, a column-request, and a common acknowledgment, these three wires support the communication. Additionally, the tree arbiter and hand-shaking are involved in this design.

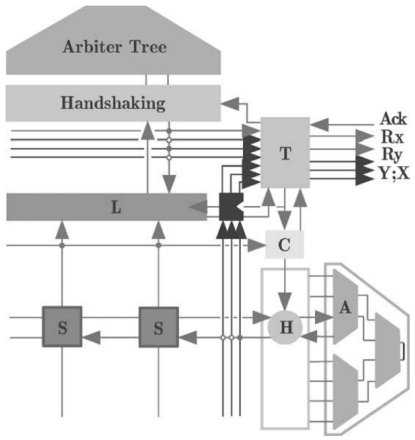


Figure 2.5: The structure of a event-driven transmitter [4]

In terms of the receiver end, [5] proposed a receiver for AER protocol, shown in figure 2.6. In this receiver, row and column addresses are used to identify recipients, although they are not transmitted simultaneously. The row address and column address are latched by $D$ and $E$, respectively. As soon as the burst finish, the address is going to be decoded and the data is written into row $R$.

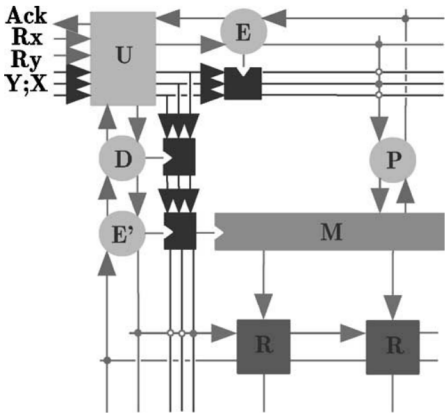


Figure 2.6: The structure of a event-driven receiver [5]

## 2.2 Communication System

Basically, the communication networks can be divided into two main categories: circuit-switched and packet switched[1]. In terms of the circuit-switched network, a virtual circuit is established when the data is propagated. Once the communication is done, the hardware resources is going to be released and converted into other virtual supporting new data communication. On the other hand, packet-switched networks work by time-multiplexing the network's component segments. Compared with the previous circuit-switched, there is no longer existing an end-to-end path. Hence, a packet has to carry enough information for each step in the communication network to identify what the packet's next step should be. In the following content, we mainly investigate the packet-switched network.

### 2.2.1 SerDes

A Serializer/Deserializer (SerDes) is a couple of functional blocks that are frequently applied in the high speed communications. Normally, the serializer regards the transmitter and the deserializer regards as the receiver during the communication. Figure 2.7 reveals the structure of a typical SerDes device. The transmitter transforms parallel data to serial format and then sends the serial data to the transmission media. By multiplying the reference clock, the PLL provides an internal high-speed serial clock for the serializer. In the aspect of receiver, it is responsible recover the clock in CDR. At the same tie, the parallel format data is restored with the help of recovered clock [6].

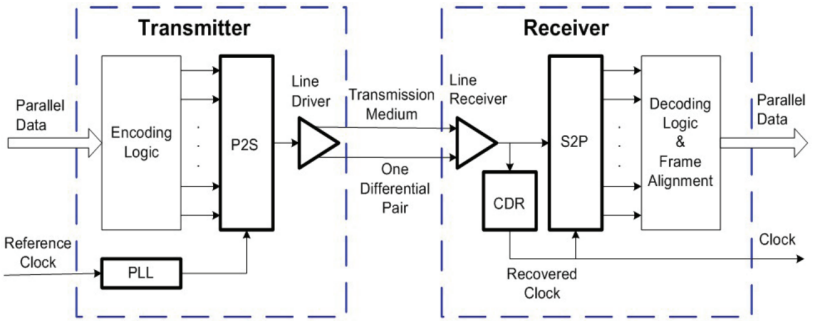


Figure 2.7: The structure of a typical SerDes device [6]

Due to the sequential information involved in the system, the power consumption is significantly occupied by the clock. Normally, the power of SerDes device is on the order of milliwatts even though there are distinctive working conditions and fabrication technologies [24] [25]. As a result, in order to eliminate the clock blocks from the communication system and reduce power consumption, the asynchronous communication link is proposed and we demonstrate some research regarding this kind of link.

### 2.2.2 Asynchronous Communication link

Compared with the synchronous communication links, it is challenging to ensure the synchronization between receiver and transmitter in the asynchronous communication links, thanks to the clock excluded in the system and Tx and Rx located at two separative clock-domain. To address this challenge, the handshake is applied to the asynchronous link. However, it leads to the asynchronous protocols becoming relatively slow due to the demand of requirement and acknowledged transition. In the following content, we introduce and compare various asynchronous links with specific synchronization mechanisms.

Based on the concept from [26], Carlos proposed a low power fast ON/OFF voltage mode communication link, which aims to be applied into the high-speed bit-serial Low Voltage Differential Signaling AER chip grids in [7]. Normally, events are asynchronous and sparse in AER systems. As a result, significant additional power savings can be realized if the links are turned off during inter-event intervals and immediately turned back on when a new event needs to be broadcast. Figure 2.8 reveals the architecture of this fast turn on/off asynchronous communication links. Once the transmitter read the AER input data, the system turns on and the circuit 4 phrase handshaking is performed before sending data out by LVDS.

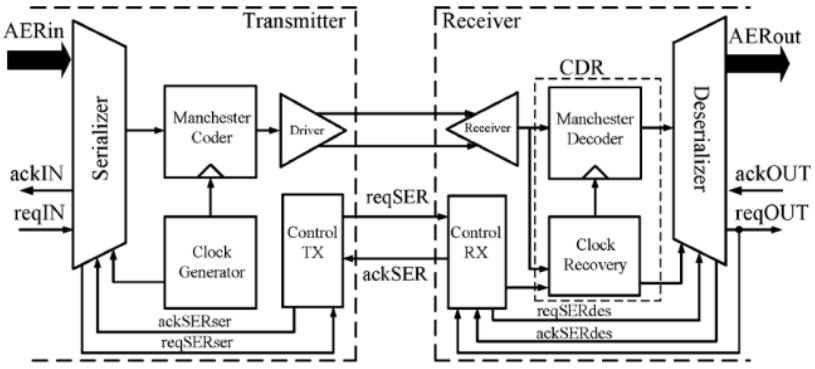


Figure 2.8: The structure of a voltage mode asynchronous links with handshaking [7]

We can notice that the reqIN leads to the toggle of reqSERser. Further, this request signal arrives at the receiver end controller. Once the ackSER signal is sent back, the handshaking done and the data starts converting into serial format. This procedure is shown in figure 2.9. It is easy to understand the handshaking mechanism consume more time and leads to the degradation of the link's throughput.

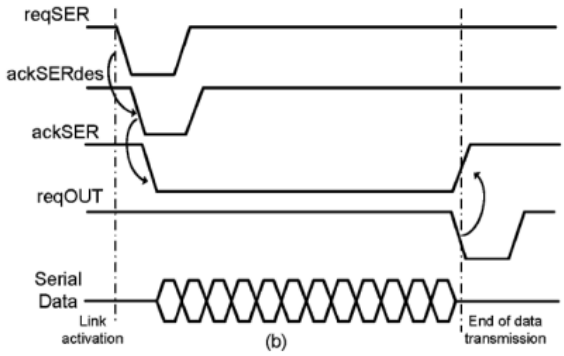


Figure 2.9: The handshaking protocol in[7]

Some research implemented the voltage mode asynchronous communication link based on the low voltage differential signal, such as [7] and [27]. However, some research revealed that the current-mode asynchronous link could be more efficient than the voltage mode, enabling almost twice longer links at the same high speed.[8]. Mochizuki[28] proposes a single-ended-style current-mode circuit with quaternary current signaling, which is used for an energy-efficient asynchronous communication link. This design achieved the throughput of 1.1Gbps per wire at the power supply of 1.2V under at 130nm CMOS technology. Compared with the traditional conventional binary dual-rail current-mode circuit[29], this design achieves the 2.3 times increase in the throughput of the link.

Figure 2.10 reveals the schematic of the current mode driver and receiver with the differential channel. In the driver end, it either blocks one of the two currents or generates two slightly different currents in the same direct. In receiver end, the input current is converted into a low voltage swing signal on output. And this design achieves the 67 Gbps finally[8].

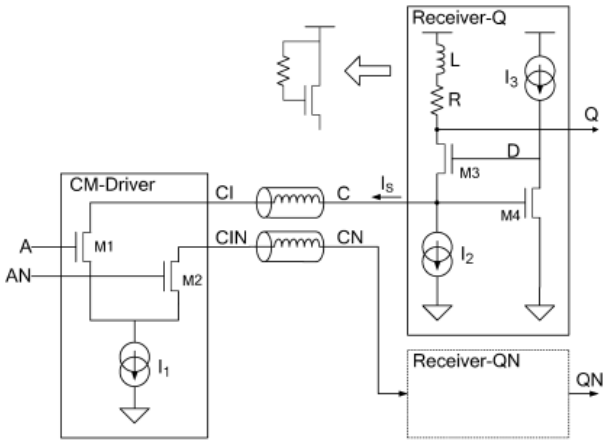


Figure 2.10: Current-mode driver, channel and receiver[8]

[9] proposed a kind of on-chip pulsed current-mode interconnect with ultra low la-

tency and the driver of this interconnect is shown in figure 2.11. Basically, the RC delay increases quadratically with the length of line. This research focus on the exploration of the inductance rather than the resistance and impedance in the wire.

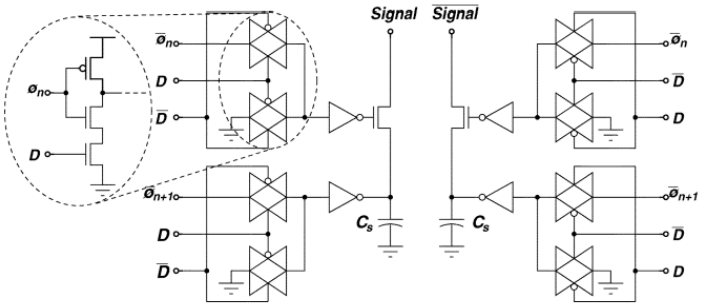


Figure 2.11: Another Current-mode driver[9]

### 2.2.3 Various Topology

There are several neuromorphic systems being implemented these years, such as the Loihi developed by Intel[12] , the TureNorth developed by IBM [11]. These designs proposed its interconnect system with various topologies. In this section, we mainly introduce how the interconnect is built in these systems.

- 2D mesh topology

  Figure 2.12 shows the TrueNorth is based on the point-to-point communication to convey a spike from one neuron to another core. Meanwhile, the hierarchical communication with a high-fanout crossbar for local communication is involved in this system. Actually, TrueNorth includes the hybrid synchronous-asynchronous flow to design interconnect elements. We mainly pay attention to the implementation of asynchronous parts.

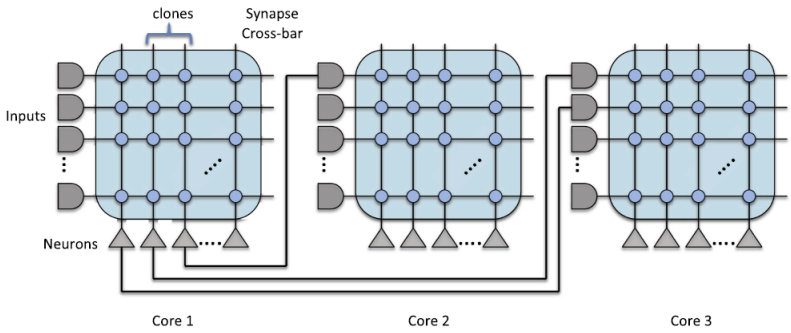


Figure 2.12: The communication system between cores in TrueNorth[10]

In the aspect of internal core, 2-D mesh network is applied to TureNorth, which is responsible for routing spike signals from neurons to axons with minimum latency. Meanwhile, some router is built to support the data travel between multiple

core inside a chip. Compared with the internal chip network, it also proposed the chip periphery, merge-split blocks and serializer/deserializer circuits, which can be regarded as the off-chip interconnect network in TrueNorth. Figure 2.13 demonstrates the top-level structure of a neural array core. We notice that the packet travel in serial format with the help of SerDes at the periphery of the core array, and packets merge/spilt before serializing/deserializing

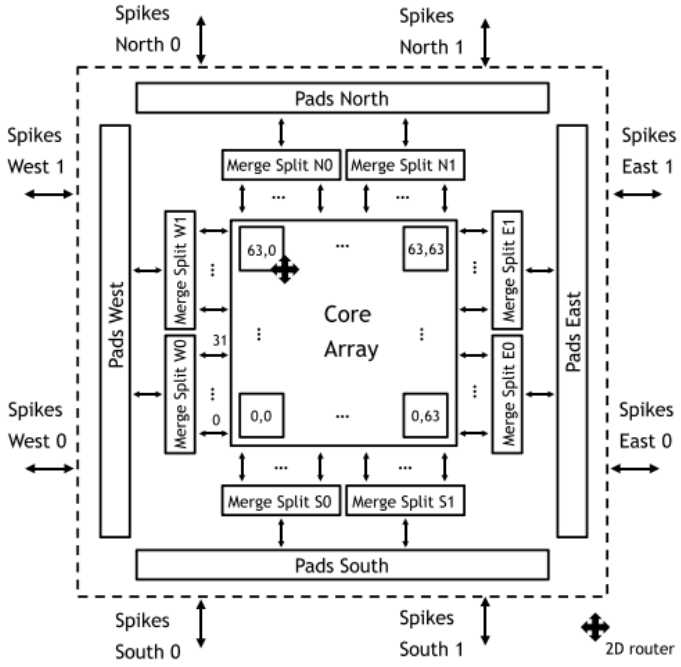


Figure 2.13: Top level blocks of the TrueNorth chip architecture[11]

Similar with the [11], a 2D mesh network cooperating with the hierarchical architecture is developed in [12]. this characteristic has the potential to greatly reduce the chip-wide connectivity and synaptic resources required to map convolutional-style networks, in which a template of synaptic connections is applied uniformly to many neurons. The total synaptic fan-in state and the total number of distribution lists, associated by axon_id is the connectivity bottleneck of this interconnect system, which are shown in figure 2.14.

- Tree topology

  Besides the mesh topology, some research proposed the tree topology network for the neuromorphic communication system. Figure 2.15 depicts a Hierarchical Neural Network Topology with a Synaptic Routing Table that depicts the connections between neurons[13]. In the meantime, each layer is made up of relay neurons(RN). An RN forwards an incoming source event to RNs at higher and/or lower levels of the TREE structure.

- Torus topology

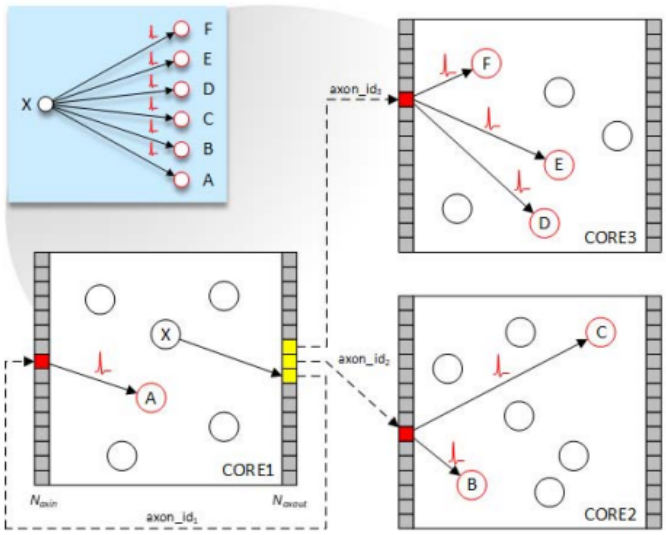


Figure 2.14: Neuron-to-neuron mesh routing model in Loihi[12]

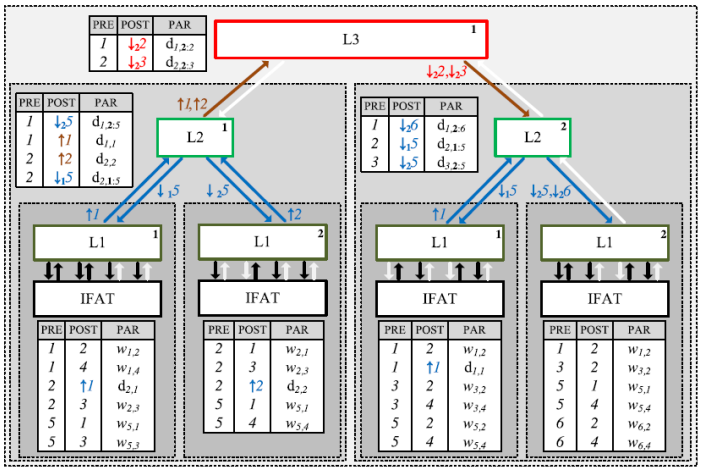


Figure 2.15: Hierarchical address-event routing (HiAER) architecture[13]

The interconnect system of SpiNNaker is inspired by mammalian brain[30] and it supports carrying tremendous packet with very small size (40 or 72 bits). Figure 2.16 reveals the topology of the system, which is different with the traditional 2D mesh network. In the aspect of inter-processor Communications There are many network interface modules responsible for converting the data from parallel format into the serial format, which can be regard as the SerDes. Besides, they are in charge of synchronizing signals that span the timing domain[14].

With the help of torus topology, every neuron in a SpiNNaker system can communicate with any other neuron via a time delay that corresponds to adjacency in biological three-dimensional space. As a result, the mapping of neurons from biological 3D space into the SpiNNaker 2D processor network can be arbitrary—every

neuron can be mapped to any processor[31].

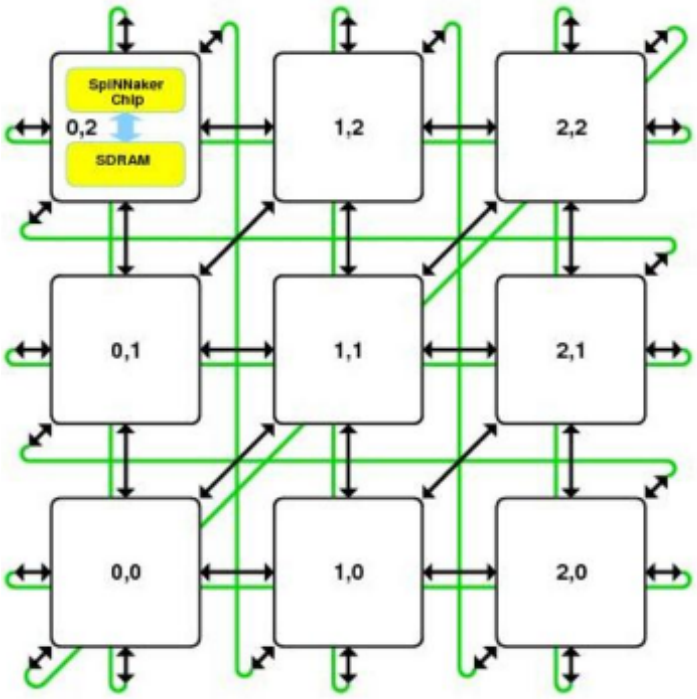


Figure 2.16: The torus topology in SpiNNaker[14]

# 3 Implementation

In the previous chapter, some research on interconnect systems has been shown. Based on existing research, in chapter 4, we will propose a new off-chip and self-timed interconnect system that is applicable to the neuromorphic computing system and elaborate on how this system is implemented from each bottom level to the top level.

## 3.1 Overview

Our work aims to build a configurable interconnect system between multi-chiplets in system-level modeling. Generally, the neuromorphic computing system includes multi-neural arrays used for accumulating spike signals. And the accumulation results are generated from the output of one neural array. Then they are sent to the input of another neural array, which can represent the data propagation between different SNN layers. In the application scenario of this work, one chiplet only contains one large neural array, which means the propagation of data is implemented by off-chip communication. Therefore, our work mainly focuses on the off-chip interconnect system.

Besides the off-chip, another main characteristic of this interconnect is self-timed. Normally the neuromorphic computing system keeps in an idle state as long as there is no spiking signal. Therefore, it is possible to build a self-timed interconnect system without a clock signal. In our system, various events are used to trigger the system to work. Due to the event-driven instead of clock-driven, power consumption experiences a significant degradation.

In figure 3.1, there is an overview of the interconnect system shown in the dash line square. We can find four chiplets, including four neural arrays, and they are all linked by the interconnect system. This system supports that each neural array can transmit the packet to any neural array, including itself, and also, the multi neural arrays can receive the packet from one origin at the same time. The system works at high speed so that it can meet the throughput requirements and avoid the data junction in the transmitter-end or receiver-end. To explain the process of implementation more clearly, we assume that the neural array always generates a spike packet with a width of 256 bits in the following chapter.

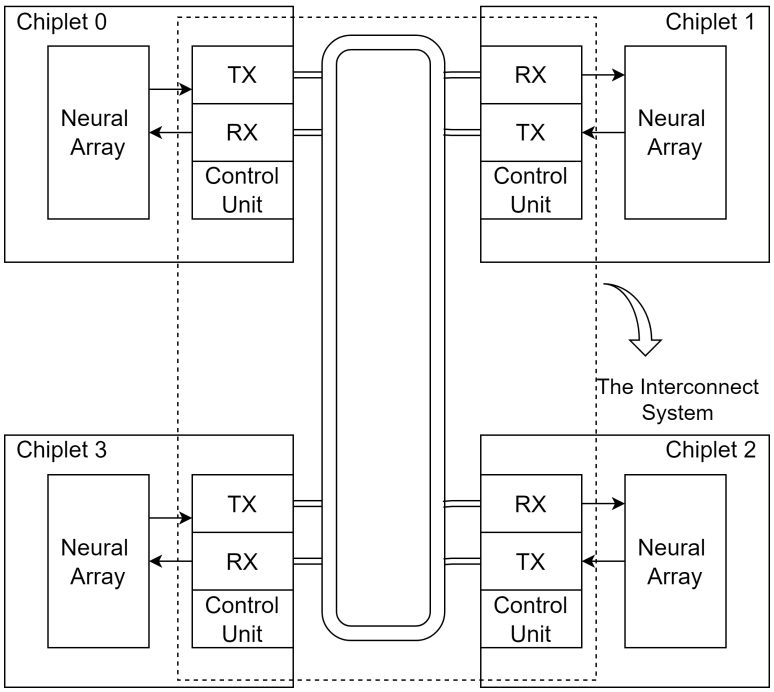


Figure 3.1: Overview of the Interconnect System

In the following chapter, we start with the basic SerDes link used for communication between one transmitter and one receiver. And then, the structure of interconnect system between multi-chiplet and the working mechanism of this system is illustrated. After that, the optimization method based on that ring topology is proposed. Furthermore, considering that the interconnect system is configurable, which parts of a system can be configured by the various parameters is illustrated. At last, we explain that the system can be configured with several parameters to prove the flexibility of our interconnect system.

## 3.2 Implementation of Basic SerDes Link

For information transmission between two neural arrays, our basic idea is to convert the parallel spike signal to serial at the transmitting end. Then the packet is sent out in serial. At the receiving end, the packet is decoded into a parallel spike signal. Considering this working mechanism, the transmitting-end can be considered as a Serializer. On the contrary, the receiving-end is considered as a deserializer. Hence, this solution is similar to SerDes link, but the clock is not involved in our system. Figure 3.2 demonstrates a SerDes Link implements the point-to-point communication between two neural arrays.

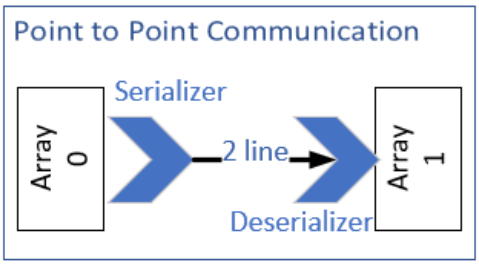


Figure 3.2: SerDes link between two neural arrays

### 3.2.1 Transmitter Design

The transmitter is responsible for sending the spike packet and indicating that packet's destination. Figure 3.3 shows the transmitter block and the input and output ports. The *packet_dstn* is used to indicate the destination of the packet. Meanwhile, considering that this subsection focuses on the basic point-to-point SerDes communication, the *packet_dstn* is only 1 bit instead of 4 bits. In terms of *enable_TX*, as long as it becomes '1', the transmitter will start converting the spike packet into serial and send out the spike packet and that packet's destination.

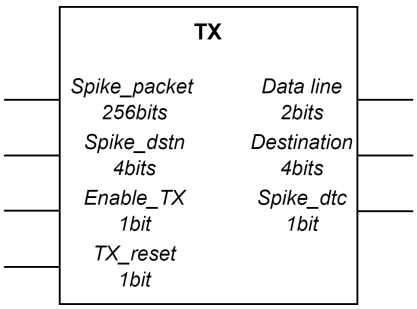


Figure 3.3: The Digital Block of Transmitter

In terms of converting the spike packet from parallel into serial, a two-wire burst-mode protocol is conducted in our system. Figure3.4 shows an example of our protocol's methodology; basically, a 256 bits packet is converted into 2 bits data lines, one line and zero line. One line corresponds to the logic '1' in the spike packet; conversely, the logic '0' from that packet is represented by a zero line. The encoding process starts from the lowest bit and ends at the highest bit of 256 bits spike packet. Consequently, some pulses present in zero line and one line with a sequence in time are used to represent the parallel signal from lower bit to upper bit. Meanwhile, no clock signals are involved in the data line since the time information has been represented in this sequential output.

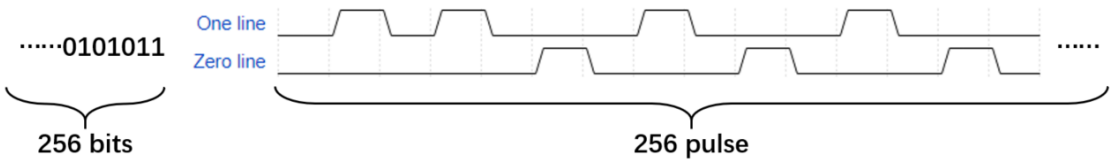


Figure 3.4: Two-wire Burst-mode Protocol

Figure 3.5 shows the structure of a transmitter to explain the working principle clearly. According to this figure, besides encoding the spike packet, we can find that the function of the transmitter also includes indicating the destination of the spike packet. Basically, the transmitter includes the following main component:

- A Pulse-mode Latch and a Normal latch: Since the data is always represented by pulse signal in SNN, it is necessary to involve a pulse-mode latch storing the data and keeping it stable temporarily. Whereas the *packet_dstn* is a level signal, a normal latch can meet the demand;
- A controller used for latch: The transmitter is an event-driven system, therefore, the latch is controlled by several events. And this function is introduced by the component *latch_ctrl*;
- Pulse Generator: This component is used to generate parallel pulse signals with specific time intervals;
- OR Gate Tree: The data from the spike packet is converted into serial with the help of this component.

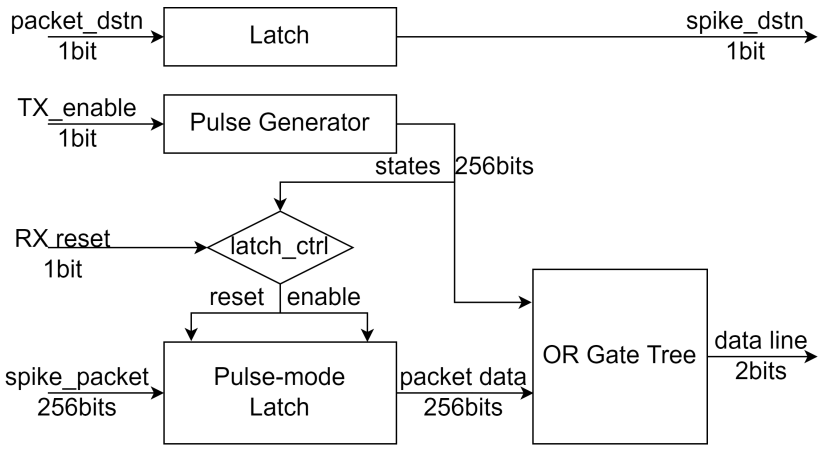


Figure 3.5: The Structure of the Transmitter

In the transmitter, the *packet_dstn* is first latched and then directly transmit without encoding as long as *enable_TX* becomes '1'. At the same time, the *spike_packet* goes

through a pulse-mode latch and is transformed into level signals *packet_data*. Meanwhile, the pulse generator starts to produce the *states*, as shown in figure 3.6. This 256 bits *states* in sequential are used to bitwise AND with *packet_data*. During this period, the controller keeps read *states* till the pulse presents in highest bit, which means the encoding is finished. Further, pulse-mode latch is reset so that it can wait for the arrival of next spike packet.

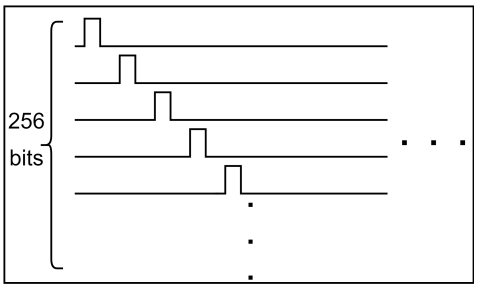


Figure 3.6: The pulse generator in transmitter

Figure 3.7 demonstrates the structure of the component OR Gate Tree. It mainly consists of two-part, bitwise AND in left side, and the gate tree in the right side. First, the bitwise AND is performed between *packet_data* from pulse-mode latch and *states* from pulse generator. After that, if logic '1' appear in *packet_data*, the pulse is allocated to corresponding *data_one*. Otherwise, the pulse is allocated to *data_zero*. Further, parallel signal *data_one* is injected into gate tree, which emerges the 256bits signal into 1 bit *one_line*. Also, the same data processing happens on *data_zero*. Basically, these 2 OR gate trees generate the 2 bits output of transmitter, and the packet data is converted into serial and sent out in this component.

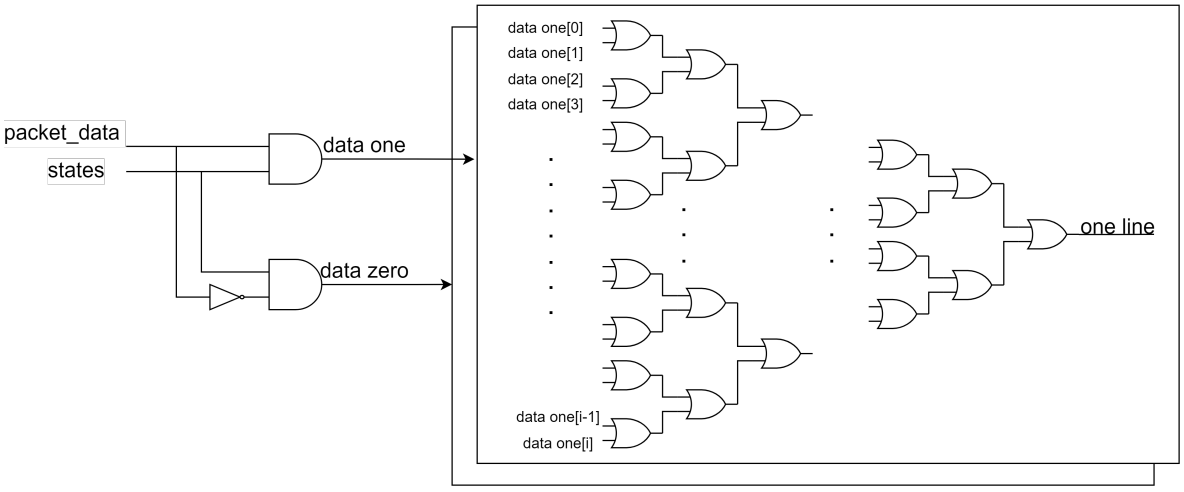


Figure 3.7: The Structure of OR gate tree in transmitter

### 3.2.2 Receiver Design

The receiver is used to decode the spike packet that is shown in fig.3.4, which means it converts the 2bits serial data line into a 256bits parallel spike packet. This process needs to be done with several components, and the following content will illustrate how to implement these step by step.

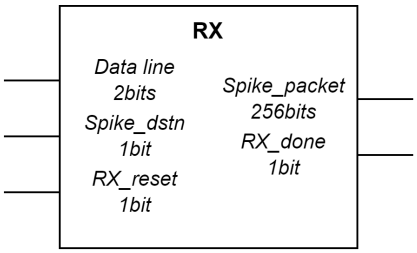


Figure 3.8: The Digital Block of Receiver

Figure 3.8 shows the digital block of the receiver. The *Data line* is connected with the same-named ports that are located in the output of the transmitter. Also, the *Spike_dstn* interacts with the same-named ports in transmitter. As long as this receiver is indicated as the packet destination, the receiver will start to decode the serial packet. After the decoding is over, RX will generate parallel spike packets at the output, and at the same time, *RX_done* jumps to 1 to indicate the finish of this conversion. Also, this signal cooperates with the controller, which will be illustrated in the following subsection.

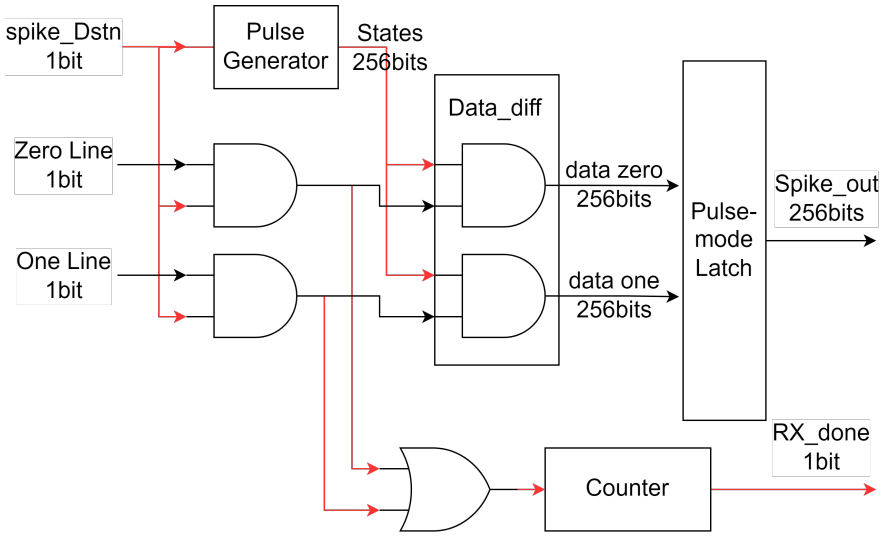


Figure 3.9: The Structure of Receiver

The receiver consists of four main components, which are an 8-bits counter, a data_diff, and a pulse-mode latch, and a pulse generator. Figure 3.9 shows how these four components collaborate. Meanwhile, the red arrow and black arrow correspond to

the control signal and data signal in this figure, respectively. In the following part, the function of each component is introduced.

- *Counter*

  In this receiver, the counter is used to count how many pulses are input. It keeps working when the pulses are injected into the clock port. When it counts to the maximum value, this counter writes a pulse in the port *RX_done*. The rising edge of that pulse means a whole packet is received and the falling edge of that pulse indicates decoding is successful. Therefore, this counter plays a 'monitor' role in receiver.

- *Data_diff*

  In figure 3.10, the *data_diff* aims to convert the data line into parallel and also differentiate the pulse from one line or zero line. During the counting period, the *states*, being from the pulse generator with the same function shown in figure 3.6 , are performed logic AND with the data line input in component data_diff. To explain the detailed structure of data_diff, the structure of this block is shown in figure 3.10. By the AND operand, the pulse from the data line can be distinguished as logic '1' or logic '0'.

- *Pules Generator and Latch*

  The pulse generator just follows the same design in transmitter, we can find the waveform of signal *states* in figure 3.6. In term of receiver's latch, it produces the parallel results one bit by one bit, which depends on whether 1 appears in data one or data zero. Finally, the decoded result *spike_out* is obtained in parallel.

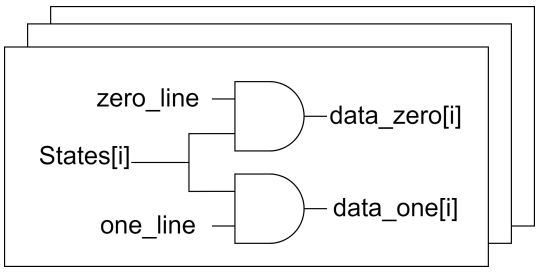


Figure 3.10: The Structure of *Data_diff*

### 3.2.3 Cooperation Mechanism

It is essential to avoid missing the spike packet during transmission, so we must ensure the transmitter and receiver cooperate correctly. The feedforward control is introduced in our basic SerDes link. In order to achieve feed-froward control, the on-chip delay and off-chip delay cooperate with the transmitter and the receiver.

In terms of on-chip delay, according to figure 3.11, we can find the *spike_dstn* is always sent out before the spike packet since the delay has cooperated with the latch and

combinational logic circuit in the datapath of the spike packet. Whereas, in datapath of *spike_dstn*, the delay has only cooperated with latch.

Meanwhile, the *spike_dstn* and *data_line* almost experience the same off-chip delay. In figure 3.9, the *spike_dstn* not only indicates the destination of a packet but also does perform the enable signal for the receiver. Only if the *spike_dstn* keeps logic '1', the receiver can read the pulse from zero line and one line and decode the packet. Otherwise, the receiver just keeps working in an idle state.

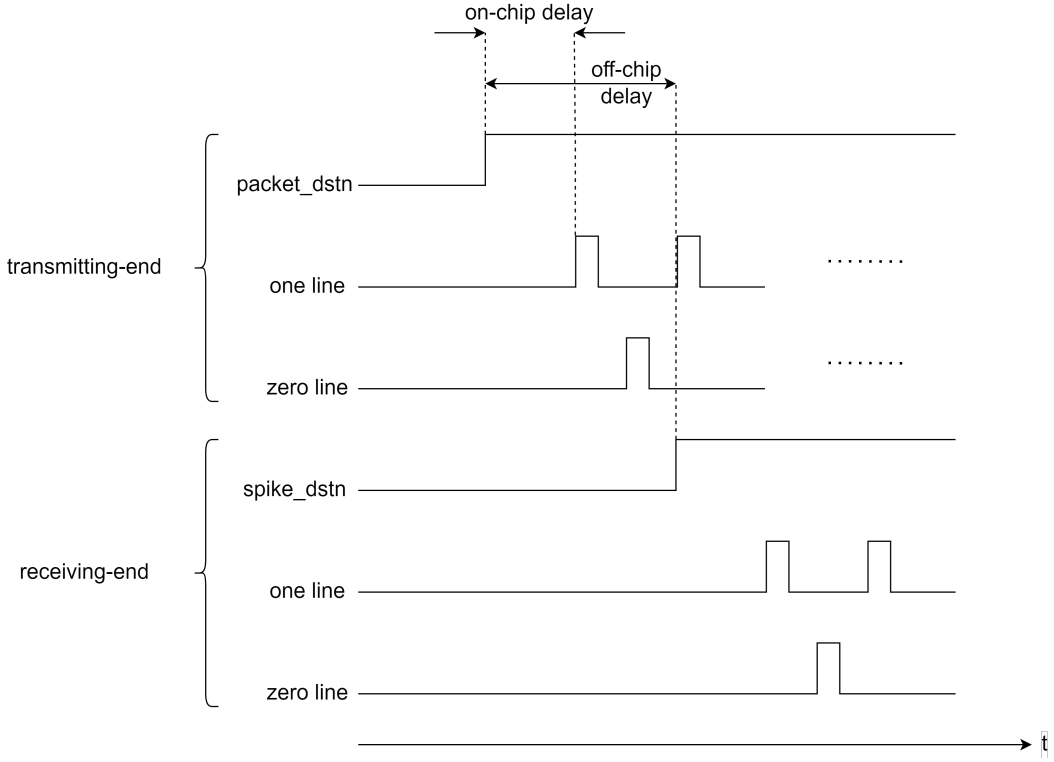


Figure 3.11: The feed-forward mechanism

As we mentioned before, the *spike_dstn* is always sent out before the spike packet, and they are going to experience the same delay in off-chip interconnect. Therefore, we can conclude that the *spike_dstn* does always arrive in the receiving-end before the spike packet, and the receiver will be ready before the packet's arrival. This feed-forward mechanism can guarantee that the transmitter and receiver cooperate correctly.

## 3.3 Implementation of Multi-Point Communication

A SerDes link has implemented the basic point-to-point communication as mentioned in Section 3.2. However, several neural arrays always cooperate in the actual use case. Hence, based on the previous SerDes link, we propose a new off-chip interconnect system with ring topology that supports multi-point communication in this chapter.

Multi-point communication means the transmission of spike packets is between several neural arrays. Besides, multiple neural arrays sometimes intend to send data

simultaneously. For this situation, the control system is introduced to our interconnect system to avoid data conflict during transmission. On the other hand, inspired by the literature, we propose a ring topology that can integrate multiple neural arrays. The following sub-chapter starts with the system topology and then demonstrates various detailed implementations.

### 3.3.1 System's Ring Topology

The interconnect system was developed based on a ring bus that connects four chiplets with each other and supports the packet propagates between four chiplets. Compared with figure 3.1, figure 3.12 shows more detail about the system's structure. From this figure, some arrow is shown in the ring bus, which means the ring is one-directional and the packets always travel in clockwise. Inside one chiplet, the interconnect system is comprised of several components, which include a transmitter, a receiver, and a local controller and corresponding multiplexer. The collaboration between these components supports the packet transmission properly.

Also, we can find there are two ring buses that represent control flow and data flow, which are shown by a blue line and a red line, respectively. The composition of these two rings are:

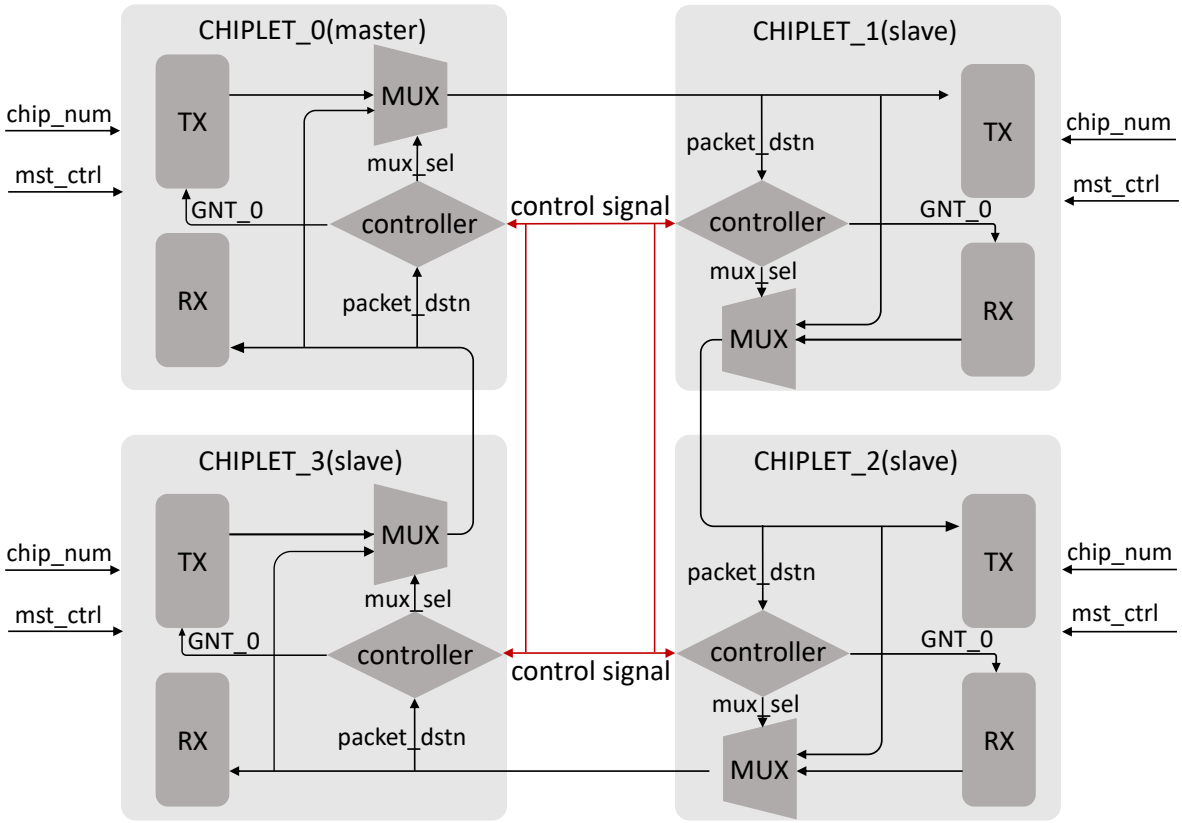


Figure 3.12: Interconnect System with the Ring Topology

- Data flow:

  The data signal is composed of the serial spike packet and also the destination of that packet. There is a 6 bits signal, of which 4 bits are allocated to the destination signal, and the other two bits are allocated to spike packet. Since any one or multiple chiplets could be the destination at once transmission, it requires at least a 4bits signal to represent the destination information. In terms of the data signal, it follows the same design that is shown in figure 3.4.

- Control flow:

  The control signal is interacted with all components in the interconnect system and it consists of the 4 bits *Grant*, and *Request* and so on. Basically, this part is used to keep multiple chiplets cooperating properly. More details about how the control signal work with the controller is introduced in chapter 3.3.3.

Some multiplexers are incorporated in the ring bus, according to to figure 3.12. The reason is that sometimes the packet needs to be injected from a chiplet into the ring bus, and sometimes it just passes by the chiplet. The MUX is used to select the path according to the propagation path of the packet.

To clarify the implementation of the whole system, we can zoom in on one chiplet in figure 3.12. More detailed information inside one chiplet is shown in figure 3.13. The control flow is defined by a red line, and the data flow is defined by a double-wire arrow. Compared with the TX block in figure 3.3 and RX block in figure 3.8, the TX and RX obtain more input and output ports, which means some update happened in these two components. In chapter 3.3.2 and chapter 3.3.2, we introduce what kind of change happened to the transmitter and receiver and how they collaborate with control systems.

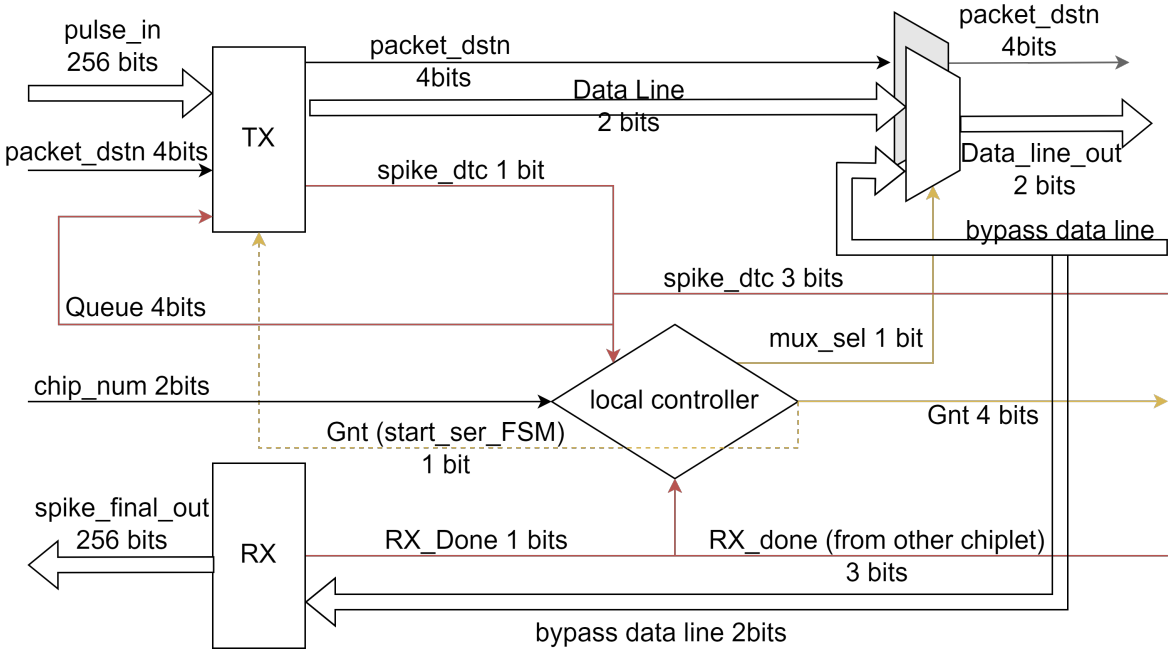


Figure 3.13: The structure of interconnect system inside one chiplet

### 3.3.2 Upgrading in Transmitter and Receiver

A transmitter and receiver with a new function are developed and follow the design shown in figure 3.14. This transmitter and receiver still focus on encoding and decoding the spike packet, but other functions are implemented to generate events driving the collaboration between the TX, RX, and control system. In the following, it will be explained separately from the two aspects of TX and RX.

- Transmitter

  The new function of the transmitter aims to achieve the hand-shaking between the transmitter and controller. And the structure of the re-designed transmitter is shown in figure 3.14.

  In figure 3.14, signal *spike_dtc* is an output from the pulse-mode latch and aims to detect if there is any bit that becomes logic '1' in the latched result. As soon as a spike is detected, *spike_dtc* becomes logic '1' and this event is sent to the local controller to inform that a packet awaits the transmission in the TX. In fact, the *spike_dtc* also works as the request signal trying to obtain permission from the controller. If TX is authorized to transmit the packet by the controller, a *enable_TX* with value '1' is received by the transmitter, which means it gets the grant and will start the transmission of this packet.

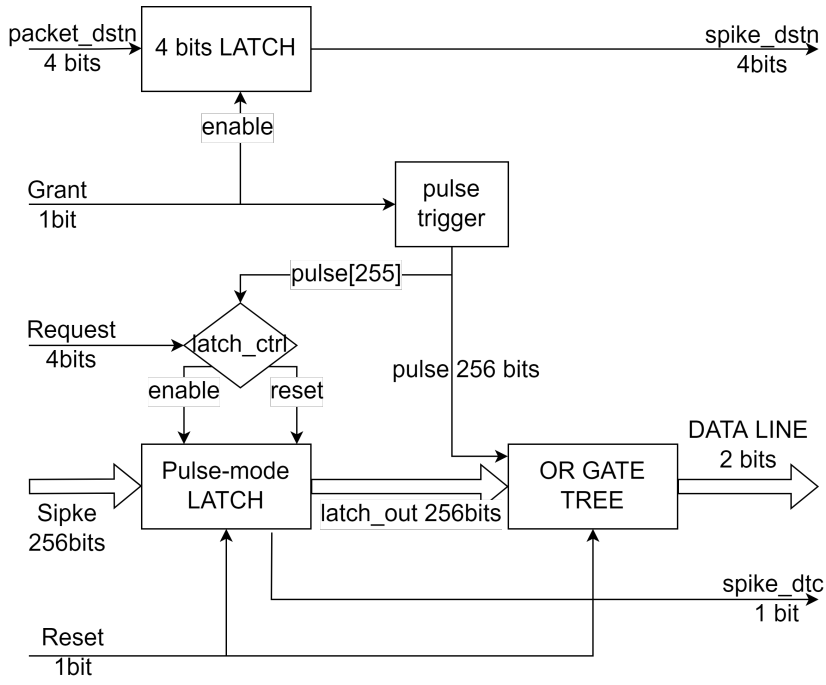


Figure 3.14: The Structure of Transmitter

- Receiver

  The new function of the receiver aims to achieve decoding of the destination signal. Figure 3.15 demonstrates the structure of this receiver. In the input ports,

the *packet_dstn* changes to 4 bits in order to support communication between 4 chiplets. Also, this signal cooperates with the *chip_num* to figure out if the receiver is enabled. For example, if the *chip_num* is equal to 0, which means the dstn_decode capture the lowest bit in *packet_dstn* and this bit is used to control whether the receiver starts converting packet from serial to parallel. Other parts of receive just follow the same design mentioned in chapter 3.2.2

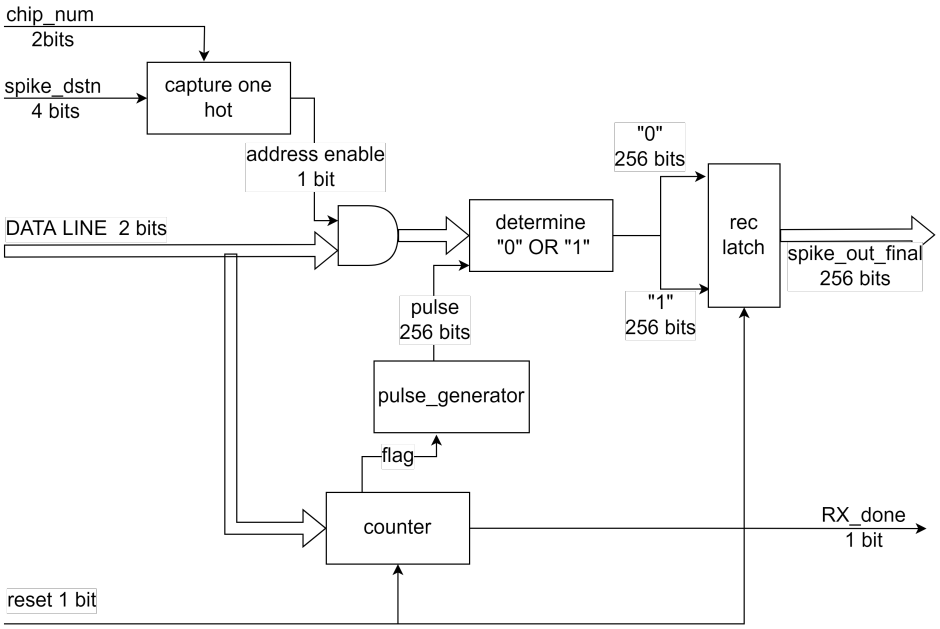


Figure 3.15: The Structure of Receiver

### 3.3.3 Distributed Control System

Ensuring that only one packet propagates along the required path on the ring bus is the main problem faced by the control system when there are multiple neural arrays that want to send packets simultaneously. At the same time, if we only develop one controller to control four transmitters, this controller is located at one chiplet, and some latency fluctuations happen on different control flows.

In order to solve the above problems, the distributed control system is introduced to our interconnect system. The distributed control system involves 4 local controllers that are allocated for 4 chiplets, and these 4 local controllers follow the same design. Furthermore, these four local controllers can be configured as one master controller and three slave controllers. Basically, the master controller is responsible for the whole system's functions, and the other controllers are slave controllers following the master controller's instructions. This architecture is shown in figure 3.16.

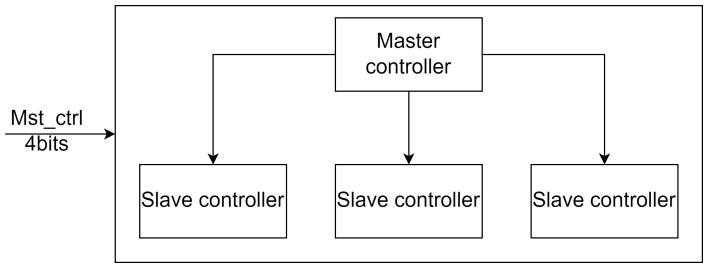


Figure 3.16: The Overview for distributed control system

The distributed control system is configured by the signal *mst_ctrl* so that we can determine which local controller works as the master controller, as the figure 3.16 shows. The implementation of one local controller can be divided into three parts, which are

- Finite State Machine(FSM)
- Round-robin Arbiter(RR arbiter)
- Bus Controller

In figure 3.17, the structure and IO ports of the local controller are demonstrated. Inside the local controller, the main part is FSM, which is responsible for the enable signal for the bus controller and RR arbiter. From the output end of the system, we can find that, basically, the system is to implement the arbitration of the request signal and to control the multiplexer to ensure that packet propagates with the specified path.

In terms of the input ports, some parameters need to be configured before starting working, which includes the *chip_num* and *mst_ctrl*. These two parameters let the system know if they belong to the master controller or slave controller, and their own location. Furthermore, the local controller can start work with the FSM. Besides the input ports mentioned above, there are three other ports relative to the FSM and arbiter and bus controller.

In terms of the outputs port of the local controller, *grant* is used to give permission to 4 chiplet transmitting packet, *Mux_sel* is connected to the 'sel' port of multiplexer, which can specify the propagation path of the packet. The other output port is used to investigate the working process of FSM.

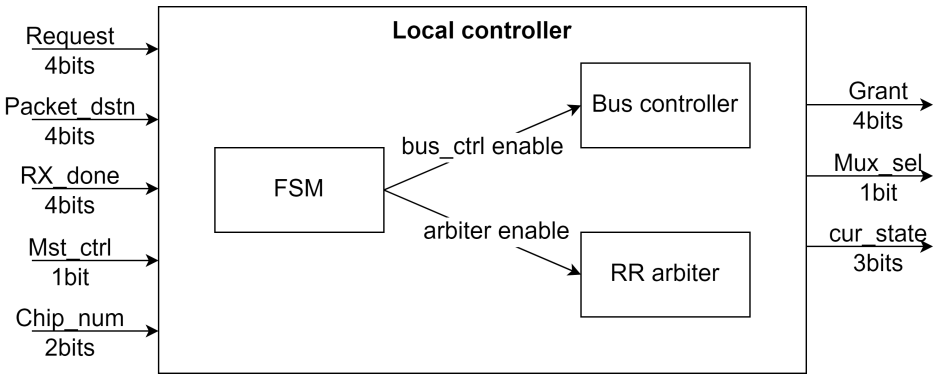


Figure 3.17: The Structure of Local Controller

The following content will illustrate how these three components work together in the local controller and the interaction between the controller and TX/RX.

- FSM

  Figure 3.18 shows the state diagram of the finite state machine. There are five states representing the working process of the local controller. Basically, the controller is defined as the master controller or slave controller depending on the value of *Mst_ctrl*. If this parameter is '1', the local controller work as the master controller. Otherwise, it works as the slave controller.

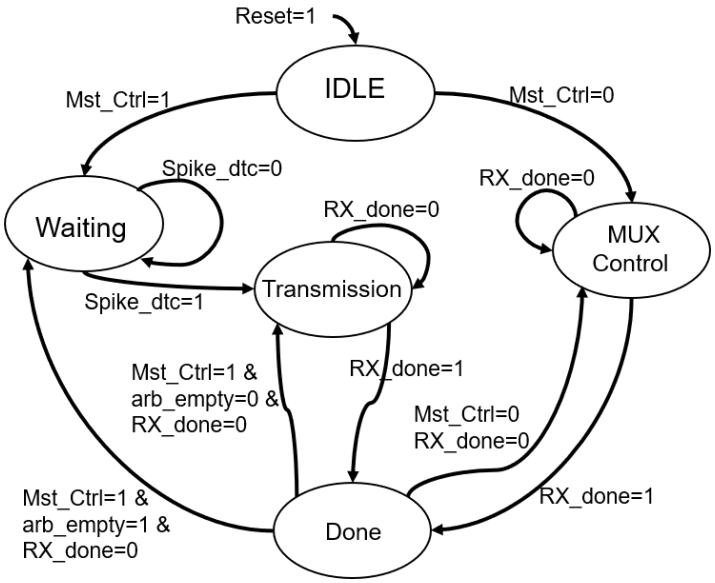


Figure 3.18: The States diagram

When the local controller is configured as the slave controller, the control logic comes into *MUX control*. In *MUX control*, the output of *bus_ctrl_enable* turns to '1' so that the bus controller is enabled and going to start calculating the MUX selection and determining the datapath of the packet according to its origin and destination. The control logic of bus controller will be illustrated in the following content. Further, if the state machine detects the *RX_done* becoming '1', it comes to *Done*, which means one spike packet has been transmitted and received. And then, the state machine returns to *MUX control* when *RX_done* becomes '0' so that it keeps ready to calculate the datapath for the next packet.

In terms of working as the master controller, the control logic starts with the 'waiting' state. Meanwhile, the *bus_ctrl_enable* and *arbiter enable* keep logic '0'. Further, the control logic works at 'transmission' state, as long as the *spike_dtc* becomes '1'. In this state, the arbiter is enabled and deal with the request signal from transmitter. Also, the bus controller is enabled so that it can arrange the specific datapath for corresponding spike packet. The control logic stays in this state until a packet is decoded by RX, which means one packet is received successfully and *RX_done* turns to '1'. And then, the control logic goes to 'Done' state

and both the arbiter and bus controller turns off. The falling edge of *RX_done* triggers the next state transition. If there is no request signal anymore, the control logic returns to 'waiting' state; otherwise, it goes to 'transmission' state and start transmitting next spike packet waiting in the Queue.

In conclusion, when the local controller works as the master, it controls the bus controller and arbiter. Otherwise, it works as a slave, which means it only controls the bus controller. Because the multi-arbiter leads to the data conflict in ring bus, only one arbiter is allowed to work. Additionally, the whole state machine is asynchronous since it is based one the event-driven without the clock signal, which means the transfer of states only trigger by an event happened.

- Bus controller

  The bus controller aims to allocate a proper datapath for the corresponding packet in the ring bus. According to the origin and the destination of the packet, a packet needs to bypass a chiplet or just end the propagation in one chiplet. Basically, the above behavior is implemented by the MUX and corresponding *Mux_sel* in each chiplet, as shown in figure 3.13. In the control logic of this component, the specific datapath is determined by three parameters, which are *chip_num* and *spike_dstn* and *origin*.

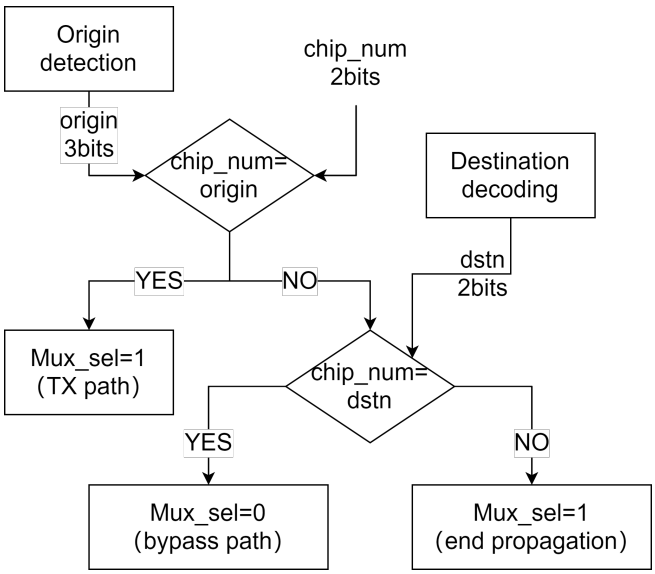


Figure 3.19: The control logic of bus controller

Figure 3.19 shows the control logic for *Mux_sel*. As long as the FSM works at 'Transmission' state or 'MUX control' state, the bus controller executes this logic and determines the result of *Mux_sel*. There are three kinds of scenario leading the result of *Mux_sel*, which are the packet starts from this chiplet or the packet bypass this chiplet or this packet is the final destination of the packet.

In principle, the *Mux_sel* turns to '1' when the specific chiplet is the origin of spike packet or the destination of the packet. Otherwise, the *Mux_sel* keeps '0' so

that the packet can bypass that chiplet. This strategy ensure the transmitter can inject the packet into ring bus properly and the packet ends at the destination avoiding energy loss.

- Arbiter

  It is essential to involve an arbiter in the control system, when the multiple chiplets require to send the spike packet at the same time. Therefore, a fixed priority arbiter is implemented to achieve the arbitration among multiple chiplets. Basically, an arbiter gets the 4bits *Request* signal representing the *spike_dtc* from each chiplet and the *arbiter_enable* from FSM. Once the *arbiter_enable* becomes '1', the fixed priority arbiter starts processing *Request*. And then, the *Grant* will be generated in the output port that can represent the permission for transmitting packet. The data processing applied in input *Request* is shown in the equation 3.1. When detecting the rising edge of *arbiter enable*, the arbiter starts the following calculation obtaining the *Grant* and the updated *Request*.

$$\begin{aligned} Grant &= Request \cap \overline{Request - 1} \\ Request_{new} &= Request \cap \overline{Grant} \end{aligned} \tag{3.1}$$

  Basically, if chiplets require transmitting a packet, the corresponding bit in *Request* turns to 1. And then, the arbiter offers the permission from chiplet0 to chiplet3, once the FSM works at transmission state, which means the chiplet0 has the highest priority to transmitting data and the chiplet3 has the lowest priority.

Generally, the distributed control system guarantees that there is always one packet propagating in the ring bus and the packet following the desired path can arrived at its destination correctly. Also, latency is significantly important in this control system. Therefore, the timing information is under consideration with timing constraints.

## 3.4 Optimization

In this section, an optimization of topology is proposed. Compared with the system with ring topology explained in Section 3.3, the new topology supports packet propagation in any direction. In previous design, the unnecessary latency and energy consumption is involved in some cases, considering that the one-directional system only supports the propagation of packets on clockwise. For example, in figure 3.1, when a packet generated by chiplet 1 intends to go to chiplet 0, that packet has to go through chiplet 1 and chiplet 2, rather than going left directly, owing to the one-directional topology. Therefore, the higher latency and more energy consumption are incorporated in the previous one-directional topology, and it is meaningful to develop a bi-directional topology in this case.

### 3.4.1 Bi-directional Ring Topology

The interconnect system with bi-directional ring topology supports that the packets propagate clockwise or counterclockwise between the chiplets. The propagation path depends on the destination and origin of the packet and the distributed control system is able to choose a specific path that can decrease the latency and power consumption during the propagation.

The optimized interconnect system almost follows the same design as a one-directional ring topology, which consists of the transmitters and receivers and a distributed control system. Compared with the previous design, the difference only happens on a component of the local controller and the multiplexer of the ring bus. Meanwhile, the clockwise ring bus in figure 3.1 is substituted by a directional ring bus. The ring of data flow can support two directions of propagation, clockwise and counterclockwise. In the next section, We will explain what has changed in the system and how the new modules work to determine the direction and path of the packets.

### 3.4.2 Optimized Chiplet Design

In the previous section, we have explained the optimized topology and shown an overview of interconnect system structure. Further, we zoom in one chiplet and illustrate more detail about the new chiplet design in this section. Figure 3.20 shows the optimized chiplet design. We can find that the data transmitted from the TX side can be directly transmitted to the RX side without going through the bypass channels of other chiplets. In addition, the data on the TX side can also be sent out of the chiplet through the MUX.

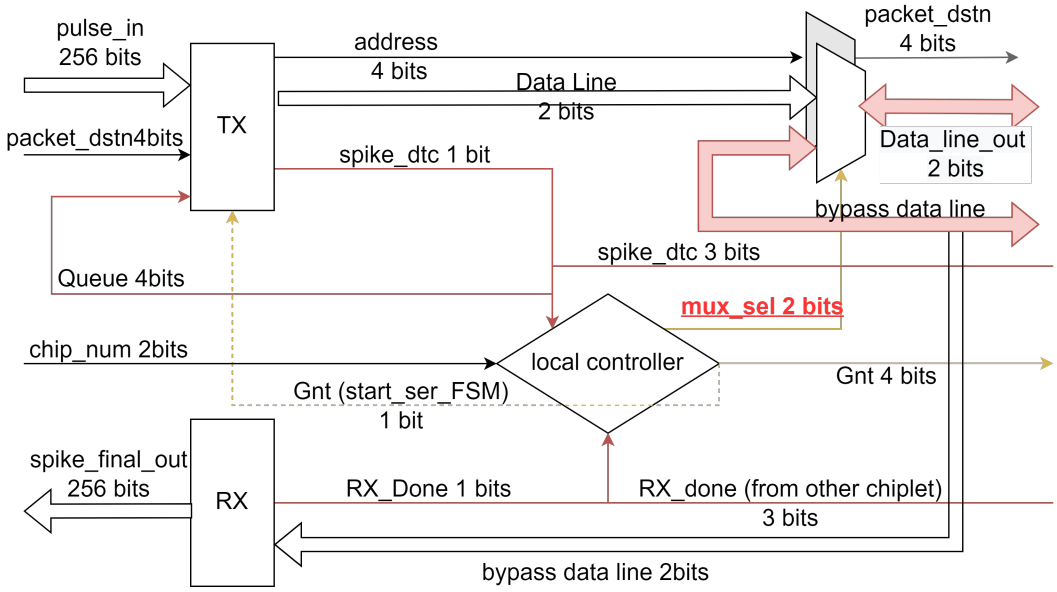


Figure 3.20: The Optimized chiplet Structure

As we mentioned before, in order to realize the function of bidirectional transmission, we mainly optimize the two modules, which are the bus controller and the multiplexer of

ring bus. In the following content, the implementation of these two parts is elaborated.

- Optimized MUX

  Normally, a multiplexer works as a data selector that supports transferring the multi-input ports to the output port based on the selection signal. While, in our design, the optimized MUX also supports transfer the data from output to input ports, which means the datapath in new MUX includes two directions. Meanwhile, the selection signal requires more bits so that it can achieve various datapath selections.

  Figure 3.21 shows the inside datapath of optimized MUX. There are 2 inout-ports that include *data line(bypass)* and *data line out*. Also, an input port is used to transfer the spike packet from TX-end. In terms of the selection signal, the *MUX_sel* becomes 2 bits, compared with the normal MUX with only 1 bit selection signal. Therefore, this selection signal can satisfy 4 kinds of datapath inside the MUX, which are represented by the arrows in figure 3.21.

  When a packet is propagated from data line (TX) to data line (bypass), it means that data is propagated counterclockwise. At this time, the data packet can be accepted at the RX side of this chiplet, or it can be transmitted to the next adjacent chiplet in the counterclockwise direction. In contrast, a packet propagates from *data line (bypass)* to *data line out*, or from *data line (TX)* to *data line out*, which means that data is propagated clockwise.

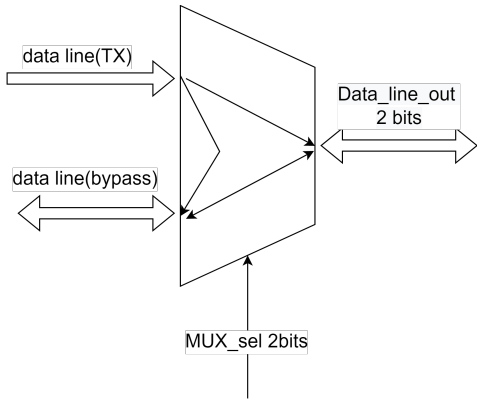


Figure 3.21: The Multiplexer with bi-direction

- Optimized Bus controller

  The Optimized bus controller is a part of the local controller and it cooperates with the FSM to control the optimized MUX. Compared with the formal bus controller in section 3.3.3, the control logic of new bus controller become more complex since there are more datapath inside the MUX. Basically, the control logic is required to determine the packet's propagating direction and if it should end propagation in this chiplet.

  Figure 3.22 illustrates the control logic of this component. According to the several parameters, the bus controller can generate the result of *MUX_sel*, which

determines the specific datapath for a packet. The upper bit of *MUX_sel* represents the packet propagation in clockwise or counterclockwise, and the lower bit of *MUX_sel* represents the packet coming from this local chiplet or another chiplets. Therefore, these two bits can fully control the various datapath in optimized MUX.

In this control logic, the packet can always reach its destination by taking the shortest path. If the starting point and the ending point are adjacent, the control logic decides in which direction propagation has the shortest path. For example, chiplet0 is the starting point and chiplet3 is the ending point, the packet can be propagated directly in the counterclockwise direction without passing through chiplet1 and chiplet2. On the other hand, when the starting and ending points of a packet are not adjacent, the path length will be the same whether it propagates clockwise or counterclockwise, so the bus controller will propagate the packet in clockwise by default.

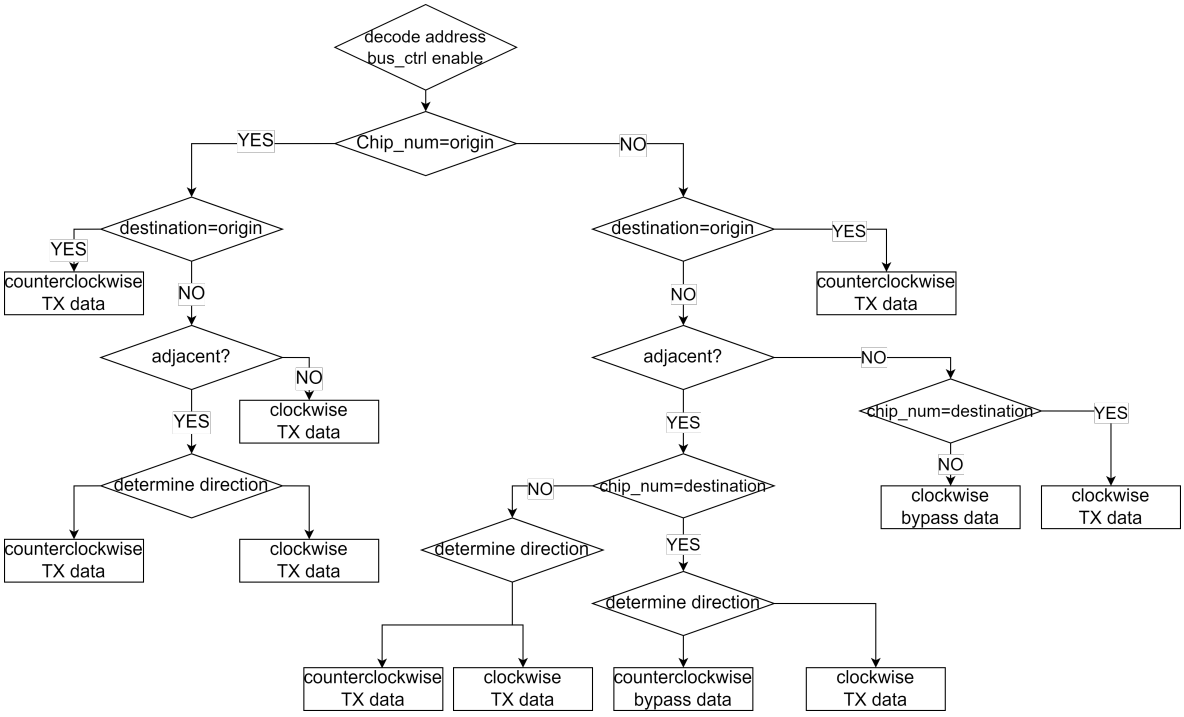


Figure 3.22: The control logic of optimized bus controller

## 3.5 Configuration and Parameterization

The implementation of the system consisting of four chiplets has been explained in the previous section. However, considering that there could be more or fewer chiplets in the application scenarios, the off-chip interconnect system should be configurable and parameterizable. Therefore, our design has the flexibility to ensure the following four parts can be configured, which are

- Distributed control system
- The dynamic destination of spike packet
- The size of spike packet
- The number of chiplet

In the aspect of the distributed control system, the configuration is supposed to be done before the system starts working. Each chiplet includes an input port named *mst_ctrl*, and we can configure the local controller as master or slave in this chiplet. No matter how many chiplets are in the system, there is only one chiplet's *mst_ctrl* being '1', which means one local controller works as the master controller, and others are considered as the slave controller.

In terms of the destination of the spike packet, it could be static or dynamic, depending on the application requirements. The chiplet can read in a static destination indicating the destination for the packet transmitting from this chiplet. Meanwhile, our system supports the dynamic destination, which means a chiplet can transmit packets with different destinations each time. This design guarantees the system supports several use case mapping into our system.

The implementation of the system remains parameterized. For the size of each packet, there is a parameter named *spike_width* to configure how many bits are in one packet. Also, the ring topology supports the interconnect system composed of a different number of chiplets. There could be either four chiplets or more chiplets connected by the ring bus. In addition, the width of *packet_dstn* is adjustable according to the number of chiplets in the system, which is used to indicate the packet's destination. In our interconnect system, the more chiplets means *packet_dstn* requires more bits to declare the destination of one packet. The following picture shows the interconnect system consisting of different numbers of chiplets.

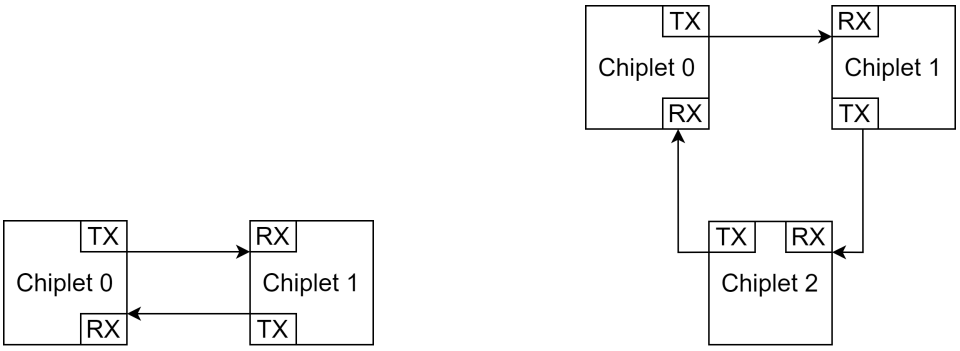


Figure 3.23: Configure the system with different number of chiplet

# 4 Simulations and Results

In this chapter, we evaluated the power, performance, and are of the interconnect system with different kinds of application scenarios. Also, we evaluated the latency and relative effect on our system so that the timing constraint is investigated. Meanwhile, the simulation environment is built and configured in cooperation with the trained MNIST network in order to verify the system's function.

## 4.1 Power and Area

The off-chip interconnect system is implemented in system-level modeling with SystemC, which means it is hard to get the accurate power and area result from synthesizing with different PVT conditions. Consequently, we did estimate the gate number from each component to the whole system. Further, the equivalent gate number combined with switching times under the specific use case can inform the power estimation result. The following content is divided into two parts. which show the results of the area estimation and the power estimation, respectively.

- Area

  The estimation of the system's area is based on the architecture of each component. Also, the change in the size of packets leads to the different structures of the transmitter. For example, the width of the latch and the number of levels in the OR Gate Tree, as mentioned in figure 3.7 are adjustable according to the width of one spike packet. In order to keep consistency with the previous section, when calculating the gate number, there are four chiplets in our interconnect system and the packet width keeps 256 bits.

  In addition, the gate categories are divided into two groups, which are normal logic gates and pulse-mode logic gates. The family of pulse-mode logic gates has been implemented in the transistor-level by the previous contributor, which includes the basic delay element, Pulse AND, Pulse OR and Pulse Latch. The reason why we involve this logic gate family in our design is that the transmitter and receiver interact with the spike signals that always keep a narrow width and the normal logic gate cannot support the calculation of such high-speed signals. On the other hand, it is essential to keep these two logic gates group since the significant difference in power consumption.

  Table 4.2 and Table 4.1 shows how many pulse-mode logic gates and normal logic gates are in one transmitter and one receiver when the transmitter and receiver are configured to process the 256bits spike packet. Meanwhile, the width of *destination* should be 4 bits due to the system consisting of 4 chiplets. Table 4.3 shows the gate number in the local controller when following the above configuration.

Table 4.1: The Gate number in transmitter's main components

| PACKET WIDTH=256 | Transmitter_wrapper | | | | |
|---|---|---|---|---|---|
| | latch_address | or gate tree | pulse_trigger | latch | In total |
| Pulse AND GATE | 0 | 722 | 0 | 0 | 722 |
| Pulse OR GATE | 0 | 1024 | 0 | 0 | 1024 |
| Pulse LATCH GATE | 0 | 0 | 0 | 256 | 256 |
| AND GATE | 0 | 0 | 0 | 0 | 0 |
| XOR GATE | 13 | 0 | 1 | 513 | 536 |
| MUX | 13 | 1 | 1 | 513 | 536 |
| OR GATE | 0 | 0 | 0 | 0 | 0 |
| LATCH GATE | 6 | 0 | 0 | 0 | 10 |
| DELAY ELEMENT | 0 | 0 | 256 | 0 | 256 |

Table 4.2: The Gate number in Receiver's main components

| PACKET WIDTH=256 | Receiver_wrapper | | | | |
|---|---|---|---|---|---|
| | rec_determine | pulse_generator | rec_latch | counter | In total |
| Pulse AND GATE | 512 | 0 | 0 | 0 | 514 |
| pulse OR GATE | 0 | 0 | 0 | 256 | 256 |
| Pulse LATCH GATE | 0 | 0 | 512 | 0 | 512 |
| AND GATE | 0 | 0 | 0 | 0 | 0 |
| XOR GATE | 0 | 3 | 512 | 0 | 515 |
| MUX | 0 | 1 | 512 | 0 | 513 |
| OR GATE | 0 | 0 | 0 | 0 | 0 |
| LATCH GATE | 0 | 0 | 0 | 256 | 256 |
| DELAY ELEMENT | 0 | 256 | 0 | 0 | 256 |

Table 4.3: The Gate number of local controller

| Destination WIDTH=4 | local controller & ring bus | | | | | |
|---|---|---|---|---|---|---|
| | FSM | bus_ctrl | arbiter | mux_address | mux_data | In total |
| Pulse AND GATE | 0 | 0 | 0 | 0 | 0 | 0 |
| Pulse OR GATE | 0 | 0 | 0 | 0 | 0 | 0 |
| Pulse LATCH GATE | 0 | 0 | 0 | 0 | 0 | 0 |
| AND GATE | 0 | 1 | 12 | 0 | 0 | 13 |
| XOR GATE | 11 | 7 | 9 | 1 | 1 | 29 |
| MUX | 7 | 5 | 4 | 1 | 1 | 18 |
| OR GATE | 0 | 3 | 0 | 0 | 0 | 3 |
| Latch | 9 | 3 | 0 | 4 | 2 | 18 |
| DELAY ELEMENT | 7 | 0 | 0 | 1 | 1 | 9 |

As the above tables show, the gate number of the transmitter and receiver are significantly larger than the local controller. Therefore, the transmitter and receiver occupy main part of area in our interconnect system. On the other hand, we calculated the kilo Gate Equivalent(kGE) of the system, in order to report the

area with the corresponding process. We considered two-input drive-strength-one NAND gate as our standard logic cell, and then accumulated the results from table 4.1, table 4.2, and table 4.3 to get the total kGE for one chiplet and a system comprised of 4 chiplets. Table 4.4 illustrates the accumulation results.

Table 4.4: The Gate Counting Results with kGE

| Gate counting | | |
|---|---|---|
| Gate | One chiplet | Interconnect system |
| Pulse AND GATE | 1236 | 4944 |
| pulse OR GATE | 1280 | 5120 |
| Pulse LATCH GATE | 768 | 3072 |
| AND GATE | 13 | 52 |
| XOR GATE | 1076 | 4304 |
| MUX | 1063 | 4252 |
| OR GATE | 3 | 12 |
| D flip-flop | 280 | 1120 |
| DELAY ELEMENT | 521 | 2084 |
| kGE(kilo Gate Equivalent) | 18.041 | 72.164 |

- Power

  In the aspects of power estimation, we mainly focus on dynamic power and ignore static power, considering that dynamic power always takes over the main part of total power consumption. The first step is to define the energy consumption per switching for various logic gates. A previous contributor has implemented the pulse-mode logic gates; therefore, we referred to her work to conduct the energy consumption of those gates. In this thesis[15], when the pulse is defined with a period of 200ns, and the power supply keeps 0.8v, the energy consumption per switching of pulse-mode logic gates is shown in the table 4.5.

Table 4.5: The Energy Consumption of Each Pulse-mode Gate [15]

| Pulse-mode Gate | Energy Consumption (J) |
|---|---|
| Pulse Delay Gate | 5.6832e-15 |
| Pulse OR Gate | 6.8950e-15 |
| Pulse AND Gate | 6.1680e-15 |
| Pulse Latch Gate | 3.4652e-15 |

Besides these pulse-mode logic gates, there is also a normal logic gate family included in our system. Basically, the energy consumption of these gates is slightly smaller than the pulse-mode logic gates. We assumed the energy consumption difference between these two groups is 10% so that the energy consumption per switching for the normal gate family is obtained directly.

In order to estimate the power for the whole interconnect system, it is necessary to define a specific use case and then we can figure out the switching times for

various gates and the time periods for executing this use once. We define two kinds of use cases that reveal the power consumption respectively.

- Basic assumptions:

  The speed of off-chip link: 5GHz;
  The time interval between two sets of spike packets: 5us.

- Use case 1:

  The off-chip interconnect system consists of four chiplets and each chiplet intends to transmit one 256bits packet to its adjacent chiplets at the same time.

- Use case 2:

  The off-chip interconnect system consists of four chiplets and each chiplet intends to transmit a 256bits packet to another three chiplets at the same time. Therefore, the act of receiving the package is twelve times.

According to the above use case, we need to get the specific switching times for various gates and the time period of executing these use cases so that we can calculate the final power result. The following equations show the process of calculation. Equation 4.1 illustrates the energy consumption when TX converts a 256bits packet from parallel to serial. Correspondingly, Equation 4.2 shows the energy consumption when RX converts a 256bits packet from serial to parallel. Due to less switching happening on the local controller, we ignored the power consumption in this part.

$$E_{TX\_per\_packet} = \sum_{gate} (\text{Switching times} \times \text{Energy consumption}) \tag{4.1}$$

$$E_{RX\_per\_packet} = \sum_{gate} (\text{Switching times} \times \text{Energy consumption}) \tag{4.2}$$

Generally, the off-chip link consumes more energy than the on-chip component. According to [32], we assume that each spike needs 26pJ when propagating it the off-chip link, which is $\lambda$ in equation 4.3.$\alpha$ and $\beta$ refers to how many packets are transmitted and received by TX and RX respectively. Based on this equation, the energy consumption during the whole use case can be converted into the system power.

$$P_{total} = \frac{\alpha \times E_{TX\_per\_packet} + \beta \times E_{RX\_per\_packet} + \gamma \times \text{Packet size} \times \lambda}{Time\_Period} \tag{4.3}$$

Table 4.6: The Energy Consumption of Transmitter

| | Times of Switching | Sum | ENERGY in total (J) |
|---|---|---|---|
| Pulse AND GATE | 512 | 3.15E-12 | |
| Pulse OR GATE | 1280 | 8.82E-12 | |
| Pulse LATCH GATE | 260 | 9.01E-13 | |
| AND GATE | 0 | 0.00E+00 | |
| XOR GATE | 782 | 8.67E-12 | 1.30E-11 |
| MUX | 262 | 2.91E-12 | |
| OR GATE | 0 | 0.00E+00 | |
| LATCH | 0 | 0.00E+00 | |
| DELAY ELEMENT | 256 | 1.45E-12 | |

Table 4.7: The Energy Consumption of Receiver

| | Times of Switching | Sum | ENERGY in total (J) |
|---|---|---|---|
| Pulse AND GATE | 256 | 1.58E-12 | |
| Pulse OR GATE | 0 | 0.00E+00 | |
| Pulse LATCH GATE | 512 | 1.77E-12 | |
| AND GATE | 0 | 0.00E+00 | |
| XOR GATE | 515 | 5.71E-12 | 1.37E-11 |
| MUX | 518 | 5.74E-12 | |
| OR GATE | 0 | 0.00E+00 | |
| LATCH | 256 | 7.98E-13 | |
| DELAY ELEMENT | 256 | 1.45E-12 | |

Table 4.6 and table 4.7 show the accumulation energy result of $E_{TX\_per\_packet}$ and $E_{RX\_per\_packet}$, respectively, when they encode and decode a spike packet with 256bits. Further, the *Time_period* is another main parameter that needs to be fixed before calculating the total power. Basically, the latency is supposed to be incorporated into this parameter and more detailed information regarding how the latency is set up during the transmission is going to elaborate in section 4.2.3. The TX needs at least 51.2ns to convert a 256 bits packet, regardless of the latency. The same time is required in the receiver end. Therefore, once transmission requires 51.2ns when the system works in a totally ideal state ignoring the off-chip delay and on-chip delay at the same time.

According to [33], the off-chip delay can be set as 2ns, which means a spike requires at least 2ns to finish the off-chip propagation. However, the time interval between two sets of spike packets is significantly larger than this latency. For these two use cases, the parameter *Time_period* is supposed to be set to 5us.

The energy consumption and power of off-chip interconnect system are shown in the table 4.8. This table reveals that the power has a strong relationship with the particular use case. The power increases as more packets are delivered and as they pass through additional chiplets.

Table 4.8: The Energy consumption and Power of System

| | $\alpha$ | $\beta$ | $\gamma$ | Energy in total(J) | Power in total(mW) |
|---|---|---|---|---|---|
| Use case 1 | 4 | 4 | 8 | 2.67E-08 | 10.66 |
| Use case 2 | 12 | 12 | 24 | 1.60E-07 | 32.013 |

## 4.2 Simulation with MNIST Use Case

In this section, some data sets from MNIST training results are applied to off-chip interconnect system to conduct the functional simulation. Meanwhile, the various mapping methodologies between the MNIST data set and interconnect system are proposed. Hence, we can ensure that more functional specifications are under consideration and verified in the proposed traffic pattern. In addition, several sets of waveforms, corresponding to each traffic pattern, are shown and explained. Finally, we investigated the limitation of the system and concluded the throughput range with various application scenarios.

### 4.2.1 Mapping Methodology

The neural network trained on MNIST consists of three layers, which are an input layer, a hidden layer and an output layer, respectively. And the neuron numbers are different in this network; figure 4.1 shows its structure. We only used a quarter of the original input data from the MNIST to develop our use case. Based on this structure, we can obtain 4 sets of data with the specific timestamp, spike_in, spike_h1, spike_h2, and spike_out, corresponding to the different layers' output in a neural network.

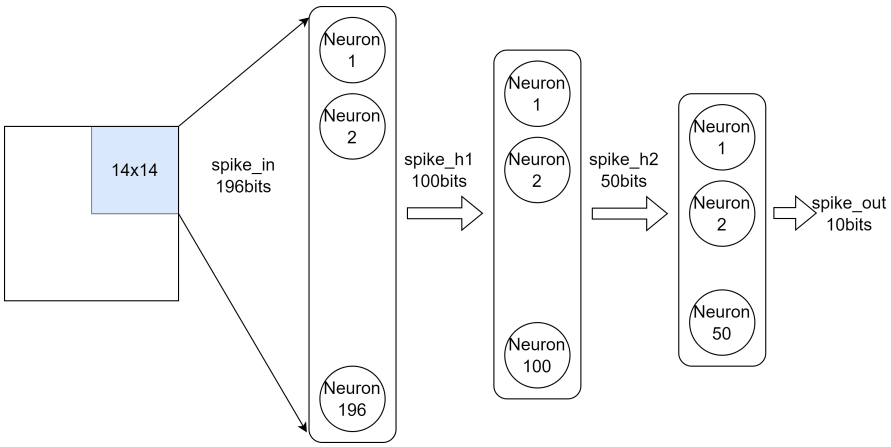


Figure 4.1: The structure of a Neural Network from Specific Use Case

Further, we aimed to map this network into our interconnect system with different patterns, and then we can carry out the simulation for the system. The network shown in figure 4.1 includes only 3 layer. However, there are four neural arrays in our interconnect system. Therefore, chiplet 0 works as a virtual input layer injecting the data into our system. Additionally, another three chiplet is responsible for acting as the input layer, hidden layer, and output layer, respectively. Since the neural array is not

implemented in this interconnect system, an interface remaining in each transmitter allows us to inject the data into itself instead of receiving data from the neural array, and this interface is shown as the red arrow in figure 4.2. Moreover, the dashed line shows the virtual data flow inside a chiplet, and the black arrow means the real data flow in our interconnect system.

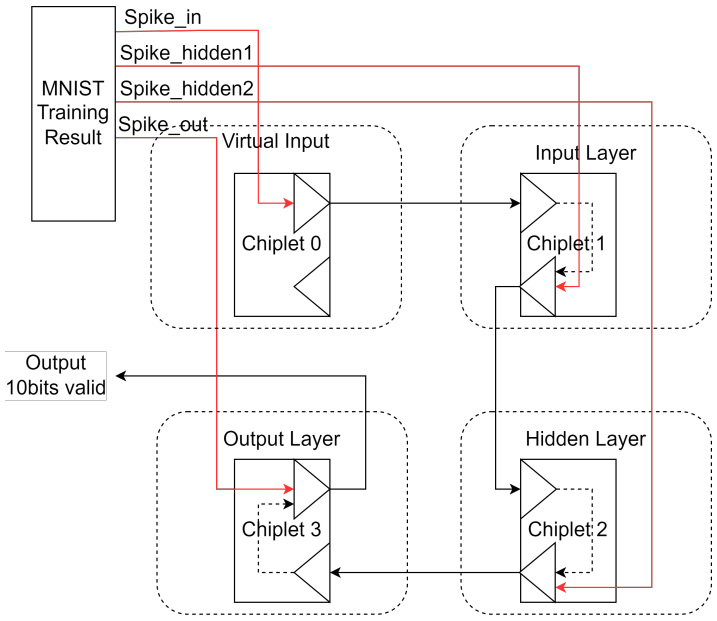


Figure 4.2: The first traffic pattern for system simulation

The second traffic pattern is shown in figure 4.3. Compared with the previous one, the situation becomes more complex in this pattern. The data Vt1 and Vt2 are combined into one spike packet, which aims to simulate reusing the neural array in one chiplet. Besides, the destination of this packet is chiplet3, which means it is attempted to bypass chiplet2 and arrived at chiplet3. Owing to this more complex traffic pattern, more functional specifications are verified.

### 4.2.2 Simulation Waveform

The whole use case, including 8560 sets of spike packets, starts from 0ms to 19.5ms and stores this information in the above four files, spike_in, spike_h1, spike_h2, and spike_out. Owing to the tremendous data under test, it is too ambiguous to show the waveform of the whole use case in one picture. Consequently, we divided the waveform into several segments and zoomed in on each segment to explain the workflow of our interconnect system.

Starting with the first traffic pattern, figure 4.4 shows one transmission between chiplet 2 and chiplet3 during this pattern. Since the master controller is located on chiplet 0, chiplet 0 also is involved in this workflow. In the following content, we explain more detail information with the waveform around the workflow shown in figure 4.4

The first waveform, figure 4.5, shows that the system boots up in the beginning stage. First, the *global_reset* becomes '1'. Meanwhile, the parameters *chip_num* and

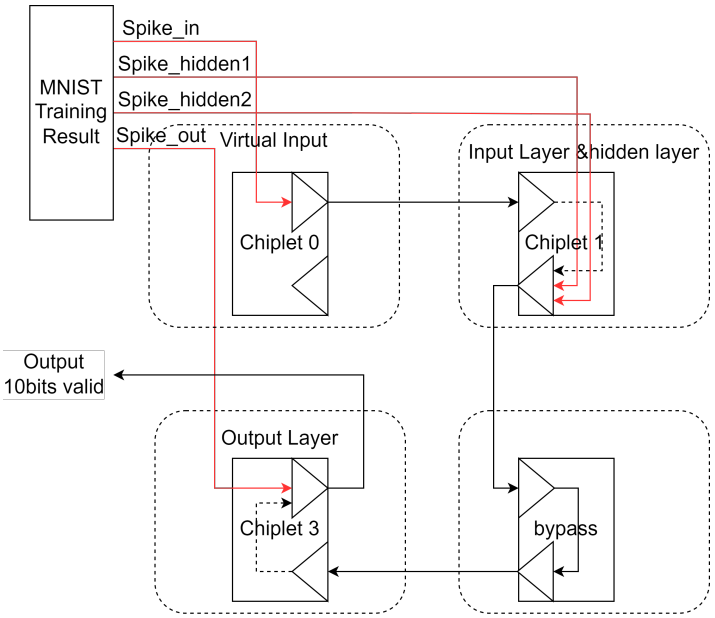


Figure 4.3: The second traffic pattern for system simulation

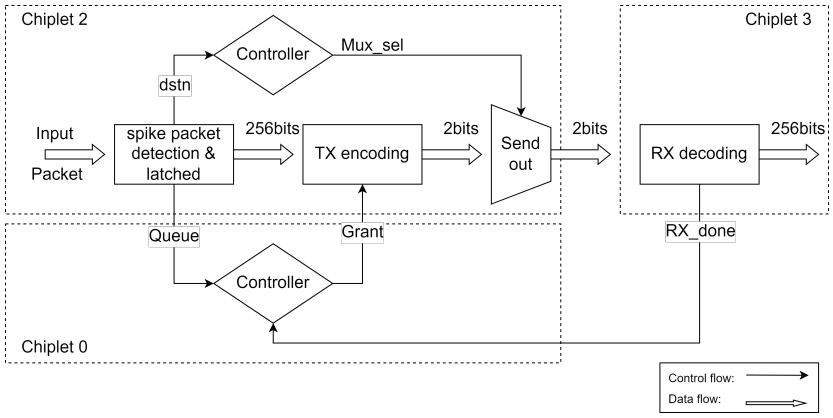


Figure 4.4: The workflow of transmitting and receiving packet

*mst_ctrl* are configured to indicate the location of each chiplet and the master controller. The controller in chiplet 0 works as the master controller since the *mst_ctrl_0* is 1. Further, the destination of the spike packet is set at 30ns in figure4.5. Since the packet always keeps the static destination, the value of *spike_dstn* is stable in the following process. So far, the system totally boots up and waits to transmit the spike packets.

Figure 4.6 shows that a packet is transmitted and leads the corresponding signals converts in the control flow. At 30ns, a spike packet is injected into the transmitter located in chiplet 2 and *spike_dtc_2* becomes 1, indicating this packet is detected. After that, TX converts it from parallel into serial. The spike packet encoding into serial is represented by *DIE2TX_one_line* and *DIE2TX_one_line*, based on the protocol in figure 3.4.

Additionally, Figure 4.6 also allows us to observe the state transformation of FSM

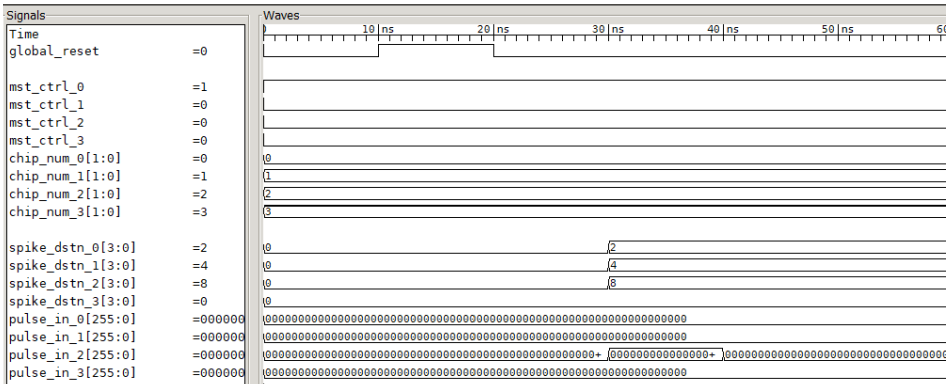


Figure 4.5: The waveform regarding the beginning stage of the interconnect system

in the local controller. According to *cur_states_0* and *cur_states_2*, the FSM belonging to chiplet 0 enters the Transmission state(010) at 31ns, indicating the transmitter is allowed to start converting packet, and the FSM belonging to chiplet2 stay in MUX control state(011) state indicating the bus controller is ready. During this period, *Queue* becomes 0100, which means a request from chiplet2 is sent to the master controller. As long as the FSM goes to Transmission state(010), the *Gnt* becomes 0100, indicating the chiplet obtains permission to send the packet.

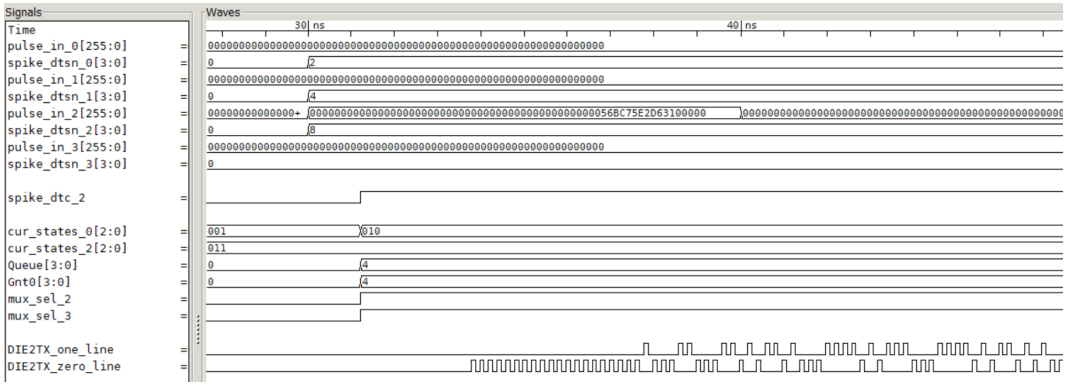


Figure 4.6: The waveform regarding transmitting a packet

Figure 4.6 shows that a packet is received successfully in chiplet3. At 86ns, the receiver generates a flag signal, *RX_done_3*, claiming the packet has been received successfully. And the final decoding result can be found in *spike_final_out_3*. Meanwhile, we can find this decoding result is fully equal to the *pulse_in_2* in figure 4.6, which means chiplet2 sends the packet to chiplet 3 successfully.

Similar to the waveform shown in figure 4.6, the working states of local control can be monitored by *cur_states_0* and *cur_states_2*. At 86ns, both master controller and slave controller enter the RX_done(100) state, which totally follows the FSM's working principle in figure 3.18. Besides, the *mux_sel_2* and *mux_sel_3* keep 1 during the transmission, ensuring the packet starts with chiplet2 and ends at chiplet3. This path-controlling mechanism can avoid the waste of power consumption.

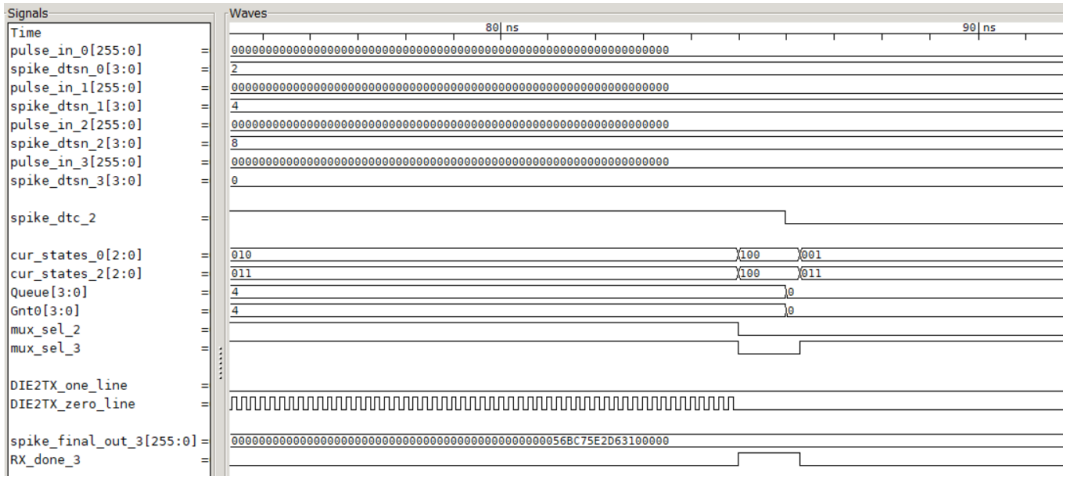


Figure 4.7: The waveform regarding receiving a packet

In those two kinds of traffic patterns, another common situation is that multiple chiplets request to transmit packets at the same time and the arbiter gives permission by specific order to accomplish the transmission. Figure 4.8 is the waveform that we obtained from the second traffic pattern. Also, this waveform shows the situation we mentioned above. Chiplet0 and chiplet1 intend to send packets simultaneously.

In figure 4.8, we can find the *pulse_in_0* and *pulse_in_1* get two spike packets and the destination of those two packets are claimed. *Spike_dtc_0* and *Spike_dtc_1* turns to 1 at the same time. Consequently, the request signal *Queue* becomes 0011, indicating two chiplet intending to get permission. The *Gnt* becomes 0001, which means TX locating in chiplet0 is granted to send packet. Further, this packet is received by RX locating in chiplet1, which is shown in *spike_final_out_1* and *RX_done_1*. As soon as a pulse is generated by *RX_done_1*, the interconnect system starts transmitting the packet in chiplet1. As a result, we can find *Gnt* becomes 0010, and the packet is sent in serial. According to *DIE1_MUX_one_line* and *DIE2_MUX_one_line*, we know that this packet start from chiplet1 and bypasses chiplet2, as the traffic shown in figure 4.3. Finally, *RX_done_3* becoming 1 indicates this packet is decoded successfully in the destination, chiplet3.

### 4.2.3 Timing constraints

Even though the whole interconnect system is self-timed and excludes the clock signal, there are still some timing constraints in the control loop. Basically, a feed-forward control is developed between the transmitter and receiver. Meanwhile, an optimized handshaking control is developed among the controller and transmitter and receiver. As long as a *require* signal is generated by transmitter and the receiver is not busy, the controller grants the transmission. 4.9 shows the related latency parameters in our interconnect system. Based on above mechanism, we propose some timing constraints to ensure these delays keep proper values and the system works correctly.

- In receiving-end: the destination signal should arrive before the data line so that

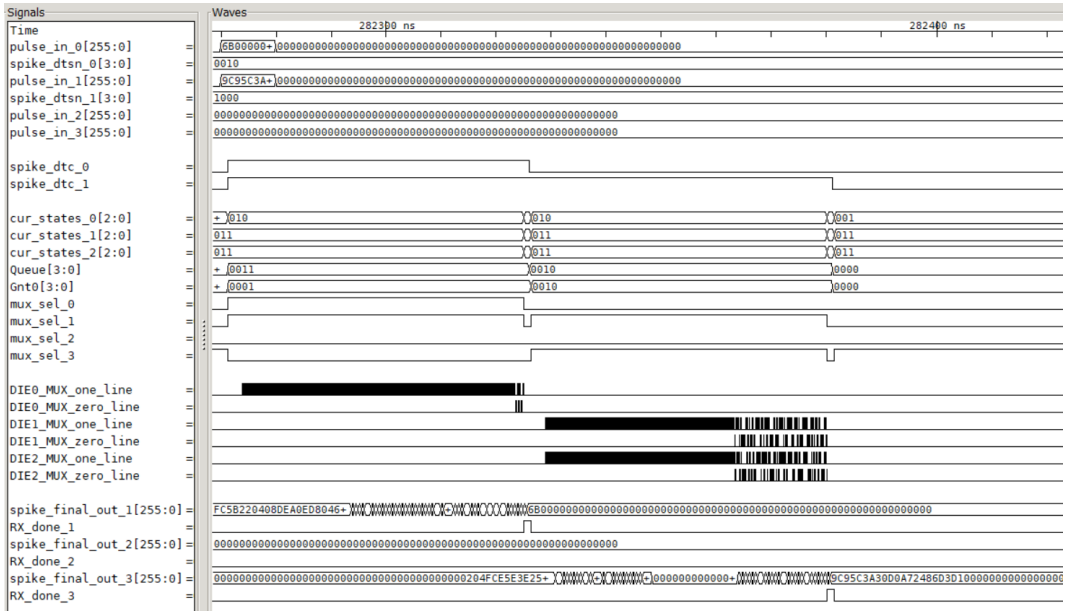


Figure 4.8: The waveform for simultaneously inserting several packets

the feed-forward is achieved.

- In transmitting-end: the spike_dtc(request) should keep stable during the transmitting period.
- In controller end: the update of spike_dtc should happens when the RX_done keeps '1'; the spike_dtc should arrive at the controller later than the spike packet latched in TX.

Table 4.9: The parameters regarding the latency of Interconnect System

| Component | Parameter | Description |
|---|---|---|
| TX | pulse_gen_delay_2 | On-chip latency in delay line |
| | tran_latch_delay | On-chip latency in latched data |
| | tran_latch_dtc_delay | On-chip latency in latched data and control logic |
| | or_gate_tree_delay | On-chip latency in combinational logic |
| Ring bus | bus_delay | Off-chip latency for data flow |
| | bus_delay_ctrl_flow | Off-chip latency for control flow |
| RX | rec_latch_delay | On-chip latency in latched data |
| | rec_dtrm_delay | On-chip latency in combinational logic |
| | rec_done | On-chip delay in counter |

### 4.2.4 Throughput Analysis

Basically, throughput is a key parameter to evaluate the performance of interconnect system. By monitoring the waveform from previous traffic patterns, the interconnect

system always works properly. And it does not meet the throughput limit during the transmission. Otherwise, some packets will miss as the interval between two packets decreases. The system's throughput is about 0.102 Gigabits per second when it works with the MNIST training results, which is significantly beyond the throughput limit. Hence, in this section, we propose various assumptions regarding traffic patterns and determine the corresponding system's throughput limit. Hence, the following content consists of two parts, which are

- The pre-defined use case

  The use cases are supposed to be defined first, and then we propose some assumptions about the on-chip latency and off-chip latency, which are crucial to evaluate the throughput. There are two kinds of use cases, which are

  - Use case 1: This use case totally follows the traffic pattern shown in figure 4.2 with the 5GHz off-chip link. In this case, each chiplet intends to transmit one 256bits packet to its adjacent chiplets at the same time.
  - Use case 2: This case aims to describe an extreme situation when we apply the one-directional interconnect system and bi-directional interconnect system, respectively. Each chiplet intends to send a packet to itself with the 5GHz off-chip link.

  In terms of latency, the on-chip and off-chip latency has been shown in table 4.9. Basically, the values of on-chip latency are on the order of nanoseconds. Meanwhile, with the help of interposer, the off-chip delay is on the order of nanoseconds as well[33]. Cooperating with the above use case, varying the values of these parameters results in different system throughput. Basically, the latency value will follow the timing constraint as we mention in section 4.2.3 when evaluating the throughput.

- The corresponding throughput limit

  When four transmitters obtained the spike packets from the neural array simultaneously, there are 4×256bits data injected into system. Based on the above use cases, we need at least 52ns to finish the transmission of one packet. Further, the throughputs with these two use cases are shown in table 4.10.

Table 4.10: The Throughput of Interconnect System

| | Throughput(Gigabits per second) |
|---|---|
| USE CASE 1 | 4.302 |
| USE CASE 2 | 3.908 |

Some verification is done in order to confirm the theoretical throughput result. In the MNIST use case, we change the timestamp and reduce its value until the interconnect system begins to lose packets. By observing the waveform from the 1st use case, some packets are missed as long as the time interval less than 238ns, which means the analysis result shown in table 4.10 is correct. The same

verification is conducted with 2nd use case, proving that the thorough result is correct.

## 4.3 Comparison

Compared with the one-directional ring topology, an optimization topology is proposed in section 3.4. In this section, we try to conclude the impact conduced by the optimized topology. Therefore, some comparisons regarding the power and performance between the two sorts of topology are introduced in the following content. The reason why we exclude the area in the comparison is that the TX and RX occupy the main area of the whole interconnect system, and the optimization that happens on the local controller does not lead to a significant difference in gate number and area as well.

The bi-directional ring topology aims to deduce the latency and energy consumption in some specific situations since the packet experiences a shorter propagation path. In order to prove the advantages of this topology, we propose one new use case to demonstrate the improvement in optimization. We assume each chiplet intends to send a packet to itself. For example, one packet with the destination 0010 is sent out from chiplet1 and received by the same chiplet. Following the same calculation methodology we mentioned above, this situation results in some improvements in bidirectional topology, which are shown in table 4.11.

Table 4.11: The Throughput and Power Comparison Between Two Topology

| | Throughput (Gigabits per second) | Power (mW) |
|---|---|---|
| One-directional ring topology | 3.908 | 13.6 |
| Bi-directional ring topology | 4.452 | 6.4 |
| Comparison result | ↑13.9% | ↓52.9% |

According to table 4.11, the bidirectional topology enhances power and performance in the given scenario. On the other hand, although we do not evaluate the corresponding area for optimized topology, there are some costs in the area. Also, the bidirectional topology leads to the degradation in configurability due to the working principle of bus controller.

# 5 Conclusion and Future Work

## 5.1 Summary

Neuromorphic computing systems always require a high-speed communication link to support the spike(packet) that travels between neurons(array). Meanwhile, due to the characteristic of event-driven in spiking neural networks, the communication link normally excludes the clock signal and related blocks. For now, many researchers developed specific solutions for communication links in the neuromorphic system. They focused on various aspects, such as the communication protocol, the structure of the network, the synchronization mechanism between transmitter and receiver, etc.

This thesis proposes a high-speed self-timed off-chip interconnect system with the ring topology for neuromorphic computing systems. Basically, the off-chip link is developed by a SerDes, which excludes the clock signal and involves the burst-mode protocol converting the packet in a two-wire serial format. And every chiplet is linked by a ring bus to propagate the spike packet. Since the fire of the neural array happens randomly, it is possible to lead the collision in the ring bus. A distributed control system is proposed to tackle this problem, which is responsible for arbitration and packet routing control. We also explored the optimization method for this interconnect system and implemented the bi-directional ring topology to improve the system throughput. Besides, the system keeps flexible and parameterizable during the design, which means it can be configured with different numbers of chiplet.

The system is implemented in high-level modeling, which means it is tough to it, but we still evaluated this interconnect system's power performance and area. The system throughput fluctuates when the packet takes different traffic patterns. But the throughput is still around 4Gbps. In order to verify the functionality of this system, a neural network from a quarter MNIST training result is mapped into our interconnect system with different traffic patterns. The latency of several components has cooperated in the simulation. Furthermore, the simulation results show the interconnect system supports the packet communication between several chiplet. Meanwhile, Under the specific scenario, the throughput of the system reaches 4.57 Gbps which can fulfill the needs of application scenarios, and the power consumption keeps 5mW with the help of pulse-mode gates in 28nm Technology.

## 5.2 Future Work

Although the performance of this interconnect system has met the application requirements, there are still some potential improvements and optimization possibilities. The following are some of the fields where more work could be done in the future:

1. Share-bus mechanism

The ring bus only supports one packet traveling on it, whatever the origin and destination of this packet. It is possible to involve the share-bus mechanism to allow multiple packets to propagate at the same time as long as there are no data conflicts. This mechanism can improve the system throughput significantly with little growth in power consumption.

2. Synthesizable

   The system is implemented in high-level modeling with SystemC. It is possible to rewrite the specific components in a more synthesizable language such as SystemVerilog. Furthermore, we can evaluate the power and area of the system more accurately.

# Bibliography


[1] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-based neuromorphic systems*. John Wiley & Sons, 2014.

[2] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain research bulletin*, vol. 50, no. 5-6, pp. 303–304, 1999.

[3] Wikipedia contributors, "Hodgkin–huxley model — Wikipedia, the free encyclopedia," 2021, [Online; accessed 13-November-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hodgkin%E2%80%93Huxley_model&oldid=1049127797

[4] K. Boahen, "A burst-mode word-serial address-event link-i: transmitter design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1269–1280, 2004.

[5] ——, "A burst-mode word-serial address-event link-ii: receiver design," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 7, pp. 1281–1291, 2004.

[6] Y. Fan and Z. Zilic, "Accelerating jitter tolerance qualification for high speed serial interfaces," in *2009 10th International Symposium on Quality Electronic Design*, 2009, pp. 360–365.

[7] C. Zamarreño-Ramos, R. Kulkarni, J. Silva-Martínez, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 1.5 ns off/on switching-time voltage-mode lvds driver/receiver pair for asynchronous aer bit-serial chip grid links with up to 40 times event-rate dependent power savings," *IEEE transactions on biomedical circuits and systems*, vol. 7, no. 5, pp. 722–731, 2013.

[8] R. Dobkin, M. Moyal, A. Kolodny, and R. Ginosar, "Asynchronous current mode serial communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1107–1117, 2010.

[9] A. P. Jose, G. Patounakis, and K. L. Shepard, "Pulsed current-mode signaling for nearly speed-of-light intrachip communication," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 772–780, 2006.

[10] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, no. 5, p. 051001, 2016.

[11] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[12] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[13] J. Park, T. Yu, S. Joshi, C. Maier, and G. Cauwenberghs, "Hierarchical address event routing for reconfigurable large-scale neuromorphic systems," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2408–2422, 2016.

[14] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. Ieee, 2008, pp. 2849–2856.

[15] F. YANG, "Designing asynchronous gate library with new system level trade-offs," 2021.

[16] J. Stuijt, M. Sifalakis, A. Yousefzadeh, and F. Corradi, "$\mu$brain: An event-driven and fully synthesizable architecture for spiking neural networks," *Frontiers in neuroscience*, vol. 15, p. 538, 2021.

[17] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2010, pp. 1947–1950.

[18] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018.

[19] M. Mahowald, *An analog VLSI system for stereoscopic vision*. Springer Science & Business Media, 1994, vol. 265.

[20] J. Feng, "Is the integrate-and-fire model good enough?—a review," *Neural networks*, vol. 14, no. 6-7, pp. 955–975, 2001.

[21] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[22] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[23] M. Mahowald, "Vlsi analogs of neuronal visual processing: a synthesis of form and function," 1992.

[24] D. Thulasiraman and J. S. Gaggatur, "A tunable, power efficient active inductor-based 20 gb/s ctle in serdes for 5g applications," *Microelectronics Journal*, vol. 95, p. 104657, 2020.

[25] M. P. Miller, F. D. Brewer, and G. Magazzu, "5gb/s radiation hard low power point to point serial link," in *2014 19th IEEE-NPSS Real Time Conference*, 2014, pp. 1–4.

[26] C. Zamarreño-Ramos, T. Serrano-Gotarredona, B. Linares-Barranco, R. Kulkarni, and J. Silva-Martinez, "Voltage mode driver for low power transmission of high speed serial aer links," in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 2433–2436.

[27] N. Qiao and G. Indiveri, "A clock-less ultra-low power bit-serial lvds link for address-event multi-chip systems," in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, 2018, pp. 93–101.

[28] A. Mochizuki, H. Shirahama, and T. Hanyu, "Design of a quaternary single-ended current-mode circuit for an energy-efficient inter-chip asynchronous communication link," in *2014 IEEE 44th International Symposium on Multiple-Valued Logic*, 2014, pp. 67–72.

[29] T. Instruments, "Lvds owner's manual," *Jan*, 2008.

[30] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[31] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[32] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[33] N. Kim, D. Wu, D. Kim, A. Rahman, and P. Wu, "Interposer design optimization for high frequency signal transmission in passive and active interposer using through silicon via (tsv)," in *2011 IEEE 61st Electronic Components and Technology Conference (ECTC)*, 2011, pp. 1160–1167.