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

CAS-2022-5035600

M.Sc. Thesis

Self-timed interconnect for SNN - From Point to Point Communication to Multi-array Segmented-bus Solution

Jinyao Zhang B.Sc.

Abstract

Spiking Neural Networks use Address Event Representation to communicate among different Neuron Arrays. To mimic the behavior of the human neural system and meets the requirement for large Neuron Array communication, the AER interconnect should be area-saving, have low power, and operates at high speed.

This thesis aims to build self-timed interconnects for point-to-point and multi-array communication. The whole system is designed at the RTL level using SystemVerilog. For point-to-point communication, two transmitters are implemented and compared according to their synthesis results. In the multi-array communication structure, we develop a generalized segmented-bus topology and the element - Fence to control its segments. Different timing problems in the design are analyzed, and corresponding solutions are proposed. The whole system can operate at around 1Gbps in a self-timed manner without any timing problems.

Self-timed interconnect for SNN - From Point to Point Communication to Multi-array Segmented-bus Solution

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Jinyao Zhang B.Sc.
born in Taiyuan, 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 **"Self-timed interconnect for SNN - From Point to Point Communication to Multi-array Segmented-bus Solution"** by **Jinyao Zhang B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 30th November, 2022

Chairman: _____

prof.dr.ir. Rene van Leuken

Advisor: _____

dr.ir. Neeraj Mandloi

Committee Members: _____

dr.ir. Chang Gao

_____

Abstract

Spiking Neural Networks use Address Event Representation to communicate among different Neuron Arrays. To mimic the behavior of the human neural system and meets the requirement for large Neuron Array communication, the AER interconnect should be area-saving, have low power, and operates at high speed.

This thesis aims to build self-timed interconnects for point-to-point and multi-array communication. The whole system is designed at the RTL level using SystemVerilog. For point-to-point communication, two transmitters are implemented and compared according to their synthesis results. In the multi-array communication structure, we develop a generalized segmented-bus topology and the element - Fence to control its segments. Different timing problems in the design are analyzed, and corresponding solutions are proposed. The whole system can operate at around 1Gbps in a self-timed manner without any timing problems.

Acknowledgments

Firstly, I would like to thank my supervisor, dr.ir. Rene van Leuken for choosing a topic that fits my interest and background and for his patient guidance during the journey of my thesis project. I would also like to thank my daily supervisor, dr. Neeraj Mandloi for his targeted advice during the project and suggestions on thesis writing. Besides, dr. David Aledo and ir. A.C. de Graaf also provide much help and guidance about my project. Lastly, I want to thank my parents for their support and care when I'm abroad.

Jinyao Zhang B.Sc.
Delft, The Netherlands
30th November, 2022

Contents

List of Figures

List of Tables

List of Acronyms

SNN	Spiking Neural Network
AER	Address Event Representation
NA	Neuron Array
MUX	Multiplexer
NoC	Network on Chip
FIFO	First Input First Output
DEMUX	Demultiplexer
FSM	Finite State Machine

1 Introduction

The Spiking Neural Networks (SNNs), as the third generation of neural networks, are driving the next wave of innovation in Neuromorphic Computing. Unlike traditional neural networks, which process data as continuous signals, SNNs handle and transmit data as discrete spikes incorporating timing and location information. At the same time, with the technology scaling, more and more neurons are integrated into a single array. Thus, a specific interconnect structure must be constructed to transmit spikes between different Neuron Arrays(NAs).

1.1 Problem statement

Neurons generate spikes as output for communication and receive spikes as input for calculation. Address Event Representation(AER) is a commonly-used strategy for point-to-point communication between NAs. While spiking activity for a single neuron is sparse compared with the working speed of electronic devices, spikes can be considerably dense in a large neuron array. In most commonly used synchronous systems, a clock consumes a lot of power which is not desired in our case. Therefore, a self-timed, high-speed, and ultra-low power interconnect for point-to-point communication needs to be designed. After that, when it comes to the scenario of multiple NAs, more requirements are added. The interconnect should be able to transmit the spikes to their corresponding destination. Besides, it must also be capable of dealing with concurrent spikes from different NAs and solving the contention among spikes.

1.2 Thesis goals

Firstly, this thesis aims to build a fully self-timed interconnect between two NAs. The interconnect should exclude the global clock signal and clock-trigger devices such as FIFOs. The interconnect should be able to deal with a spike in the order of nanoseconds(ns) and preserve the timing information of each spike as much as possible.

Then based on that, the thesis aims to build an interconnect structure of Multiple NAs. This interconnect structure uses the segmented-bus topology to save power and enable parallel communication. The NAs are separated by a structure called "Fence" which can be turned on and off. The control logic will be designed to control the whole traffic flow of the interconnect. The timing information also needs to be verified in this case.

1.3 Contributions

The main contributions of this thesis are:

- A novel self-timed point-to-point interconnect structure using AER for NAs.
- A novel interconnect architecture for multiple NAs using segmented bus topology.
- Design of the structure of "Fence" and its control logic.
- Design of the whole test environment in SystemVerilog and the simulation of the implemented structure.
- The analysis and solution of timing problems in the design.

1.4 Outline

The rest of the Thesis is organized as follows. Chapter 2 gives a brief overview of different interconnect topologies and communication schemes. Based on that, the strategy and constraints in our design are chosen. Chapter 3 gives the implementation of our self-timed interconnects structure for point-to-point communication. Then Chapter 4 describes the architecture and implementation of our multi-array interconnect solution. After that, Chapter 5 presents the simulation results of previous designs. Finally, in Chapter 6, the conclusion of the thesis is drawn, and the future aspect of the research is discussed.

2 Background

This chapter introduces the relevant background information for this thesis project. Firstly, we give a brief overview of Spiking Neural Networks (SNN) and the protocol, Address Event Representation (AER), for their inter-array communication. Then, different common implementations of AER protocol are presented and compared. Lastly, some interconnect topologies for multi-components communication systems are introduced.

2.1 Spiking Neural Networks and Address Event Representation

Spiking Neural Network (SNN) is an emerging neural network that mimics the human neural system to do processing and computing. In real neural systems, a neuron fires an action potential when its synaptic potential reaches a threshold. It will send this action potential to nearby neurons, causing other neurons' membrane potential to increase or decrease by a certain amount. The SNN works basically the same way in that it utilizes spikes to communicate among different electrical neurons, as shown in Figure 2.1. Different from traditional artificial neural networks, spikes used in SNN are discrete pulse signals and contain the timing information itself. Therefore, how to transmit these spikes between different Neuron Arrays (NAs) becomes a critical topic in the research of SNN.

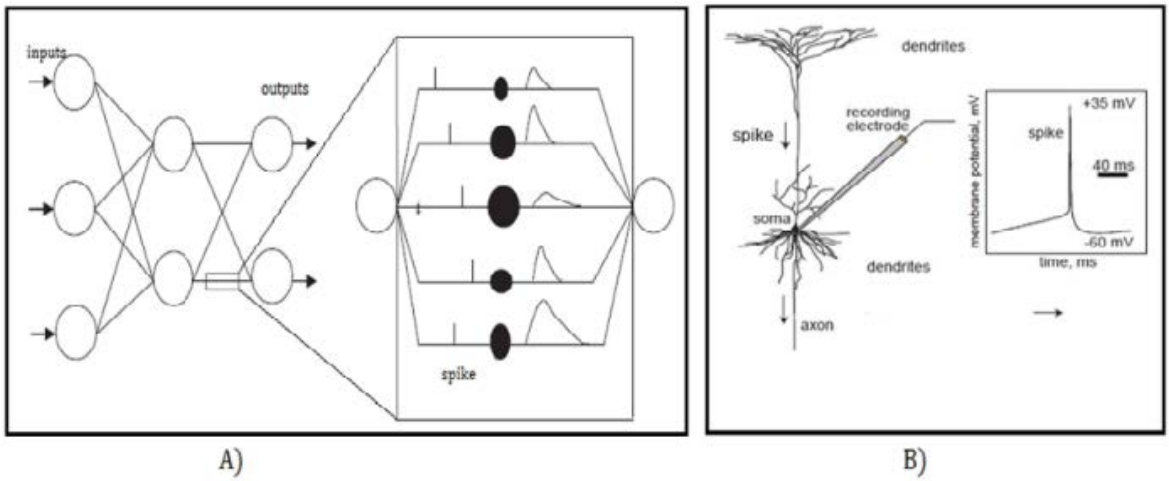


Figure 2.1: A) SNN structure[1] and B) Biological neurons[2]

Address Event Representation (AER) is a commonly used protocol for different NA communication. It was first proposed in [3] in 1993. Figure 2.2 shows a diagram of how AER works. The AER circuit from the first Neuron Array samples the spikes from different neurons and encodes them to a corresponding address. Then it sends the address to another Neuron Array. The decoder in that NA decodes the address into spikes on the corresponding address line. After that, neurons in the second array can accept these spikes as their inputs to do the following computing and processing.

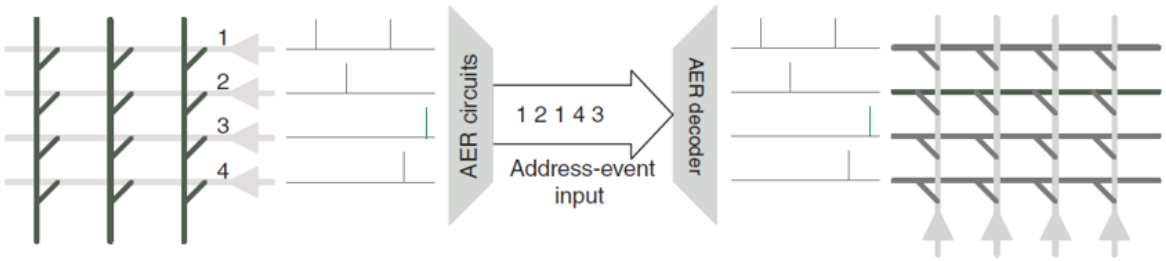


Figure 2.2: Address Event Representation [4]

The challenge of AER interconnect design falls mainly in 2 aspects from a broad perspective. Firstly, although the spiking activity for a single neuron is relatively sparse compared with modern digital circuits[5], the spikes can be dense for a large size NA. So AER circuits require low latency and high throughput. Secondly, since SNN imitates brain activity, low power is also a crucial criterion for AER design.

2.2 Communication Scheme

There are mainly 3 communication schemes for AER interconnect in terms of timing control methods. They are synchronous solutions, asynchronous solutions with handshake protocol, and self-timed circuits.

2.2.1 Synchronous Solution

Synchronous circuits are separated into several parts by edge-triggered flip-flops, as shown in Figure 2.3. Each part is driven by a global clock signal in order to function correctly. In synchronous circuits, a clock-driven feedback loop can also exist to change the next circuit state according to the circuit's current state.

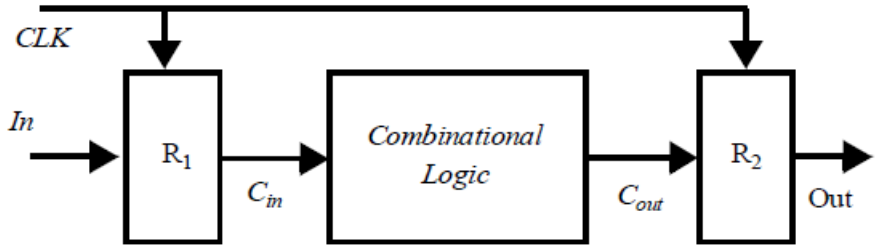


Figure 2.3: A generalized synchronous system[6]

Figure 2.4 shows a diagram of a synchronous AER transmitter proposed by [7]. The transmitter is composed of 5 blocks. The clock signal puts spikes in different cycles and pushes them into FIFO sequentially. The encoder takes spikes from FIFO and outputs an address corresponding to the spike with the highest priority each clock cycle. The feedback signal to MUX plays an important role in deciding if the encoder should handle the remaining spikes from the previous cycle or fetch a new spike set from FIFO. The whole system works in an orderly manner with the help of a clock.

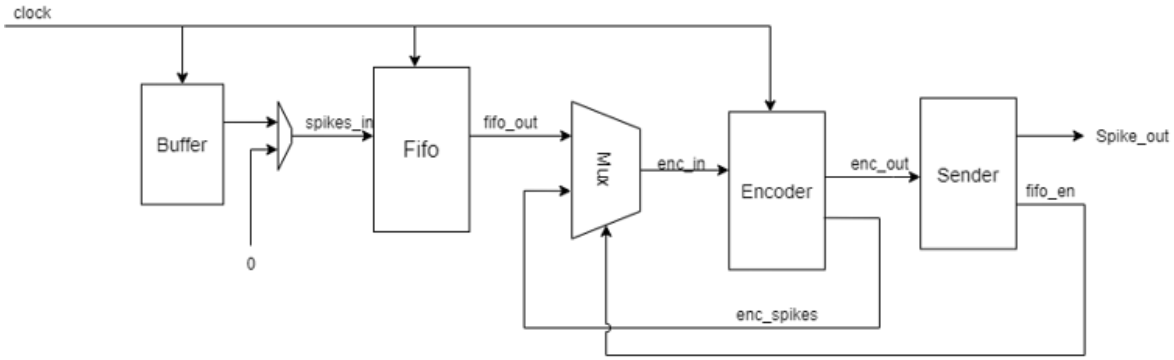


Figure 2.4: Synchronous AER implementation

Although synchronous circuits are robust and easy to implement, it also has some drawbacks. Firstly, the constantly flipping clock signal greatly contributes to power consumption. In [8], the authors point out that the global clock consumed 40 % of the chip's total power. Besides, the clock tree needs to be carefully balanced to deal with the influence that jitter and skew bring about. To handle these problems, asynchronous circuits are discussed in the next subsection.

2.2.2 Asynchronous with Handshake protocol

Asynchronous circuits do not need a global clock signal to drive the circuits. Instead, they use a pair of control signals, Request and Acknowledge, to control the data transmission. Figure 2.5 shows a basic structure of the interconnect using handshake protocol. The Request signal starts a transmission and the Acknowledge signal indicates

the transmission has been completed. Then data bus can change to start a new transmission.

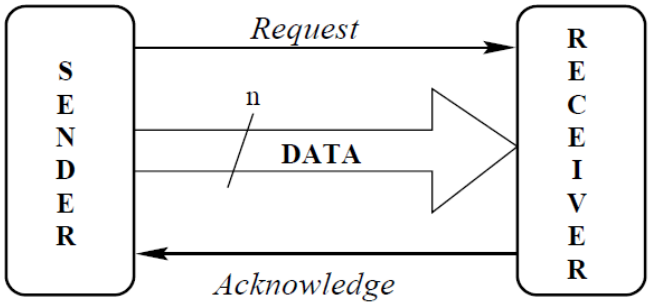


Figure 2.5: Communication scheme using handshake protocol[9]

Two kinds of handshake protocols are often used in asynchronous circuits. Figure 2.6 shows the waveform of these two protocols. In the two-phase protocol, both the rising and falling edges of the Request signal can start a transmission. This also applies to the termination of the transmission by Acknowledge signal. On the contrary, in the four-phase protocol, only the rising edge of the control signals is valid for transmission. Therefore both the Request and Acknowledge signals need to reset to 0 at the end of each transmission.

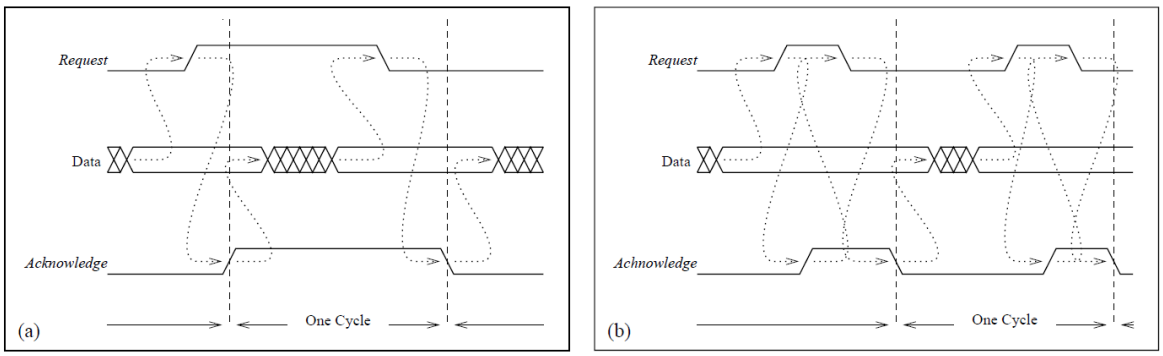


Figure 2.6: Waveform of (a) two-phase protocol and (b) four-phase protocol[9]

Most asynchronous circuits use handshake protocols to avoid the drawbacks of the global clock. But it comes at a price. The sender should be able to detect when the data is stable for transmission, and the receiver needs a completion detection circuit to generate the acknowledge signal. This will increase the area overhead. Besides, the sender and receiver must always wait for each other's control signal to continue the communication. This will slow down the communication process.

2.2.3 Self-timed circuits

Taking all the defects of the above 2 schemes into consideration, a new scheme called self-timed circuit is proposed.

There is no universal definition for the self-timed circuit. Some articles[10] mix self-timed circuits with asynchronous circuits that use handshaking. From my perspective, a self-timed circuit should have the following features.

Firstly, it should use the characteristic of the data or event itself to drive the data transmission. So it is also called event-driven or data-driven circuits. In this respect, SNNs are highly consistent with the self-timed circuit because spikes include timing information themselves. The author in [4] called this property of spikes 'time represent itself'.

Secondly, the self-timed circuit should use combinational logic, latches, flip-flops, and delay elements as basic building blocks. Delay elements are circuits that delay the input for a certain period of time. The simplest delay element is a pair of inverters.

Finally, self-timed circuits should have no clock signal, no FIFO, and no handshaking signals. They should also have few control signals, especially global ones because this could slow down the transmission.

owing to these special features of self-timed circuits, no protocol or structure applies to all kinds of self-timed circuits. Different self-timed structures need to be specifically designed for different scenarios. [11] proposes a self-timed scheme that can be used for pulse-mode circuits. It uses a series of self-resetting pulse gates to make the circuit work in order.

When it comes to SNNs, few self-timed interconnect implementations could be found. [12] proposes a self-timed interconnect for point-to-point communication. Figure 2.7 shows the structure and basic components of its proposed interconnect. It uses an input edge detector to sense the rising edge of spikes. The edge detector then drives a local multi-phase vibrator to generate 3 short pulses. These pulses then trigger the spike arbiter to output an ordered AER sequence. This design substitutes the global clock signal with a local multi-vibrator so that the pulses are only generated when there is an incoming spike. It uses the edge of the spikes to trigger the whole interconnect to work properly.

2.2.4 Delay Models of Asynchronous Circuits

Considering all kinds of asynchronous circuits, including the ones using handshake protocol and the self-timed circuits, they differ mainly in the delay models they use. The delay model refers to the assumptions and constraints imposed on the gates, wires, delay elements, and different paths of the circuit.

[13] divides the asynchronous circuits into 5 categories. They are delay-insensitive design (DI), Quasi-delay-insensitive design (QDI), Speed-independent design (SI), Scalable delay-insensitive design (SDI), and Bounded-delay design. The timing restrictions gradually increase from the first kind to the last. The delay-insensitive design makes no assumptions about the delays of all the circuit elements, while for the bounded-delay design, delay value restrictions are imposed on most of the circuit elements. As for the 2 schemes mentioned in the previous subsection, handshake circuits can be made to

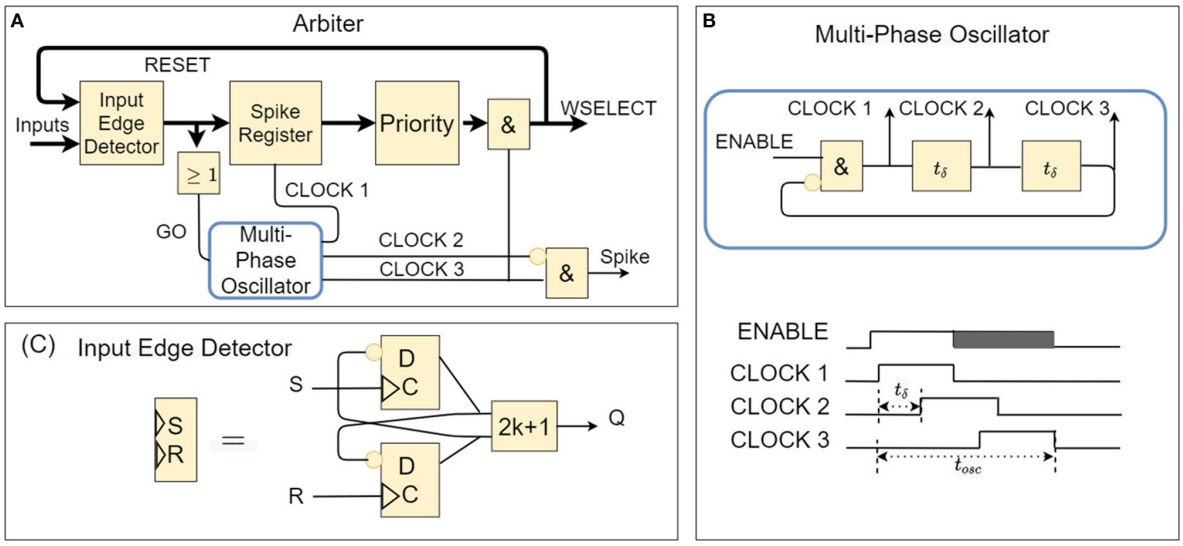


Figure 2.7: Structure and Basic components proposed in[9](a) Spike Arbiter (b) Local Multi-phase vibrator (c) Input Edge Detector

any of the DI, QDI, SI, and SDI designs based on their different variants. On the other hand, self-timed circuits fall in the category of bounded-delay designs.

Generally speaking, a design with fewer delay constraints and assumptions tends to be more robust to process variations. But this comes at a price that it may work at a relatively low speed, with the high area and power overhead. The bounded-delay designs are likely to be smaller, faster, and lower in power consumption. But the timing analysis is needed throughout the whole process of the design. Besides, this kind of circuit is hard to synthesize with the traditional flow.

2.3 Interconnect Topology

When there are multiple components in a communication system, the interconnect topology needs to be considered. In this section, 3 kinds of topology are presented and discussed.

2.3.1 Shared Bus

The shared bus is the simplest topology where all the nodes connect to a single link, as Figure 2.8 shows. Bus topology is cost-effective for a small number of nodes. But when the number of nodes increases, the contention problem becomes serious. So it is used a lot in low-frequency, low-throughput scenarios. The performance of the shared bus can be improved by increasing the number of channels[14], but this will also add extra area overhead.

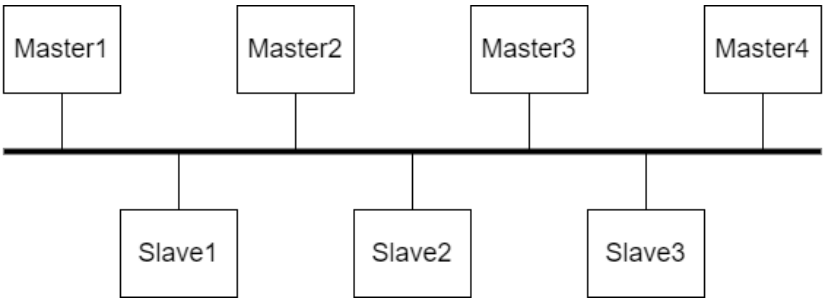


Figure 2.8: Bus structure

2.3.2 Network on Chip

Network on Chip(NoC) is an arising interconnect topology for communication systems that have a large number of nodes. Figure 2.9 exhibits the basic structure of an NoC interconnect. It is a 2D mesh that has 2 kinds of components, Processing Element (PE) and Router (R). Each Processing Element is connected to its own Router, while each Router connects directly to its adjacent Routers. To start a transmission, the Processing Element first needs to send packets to its Router, and the router determines the route the packet takes to the destination.

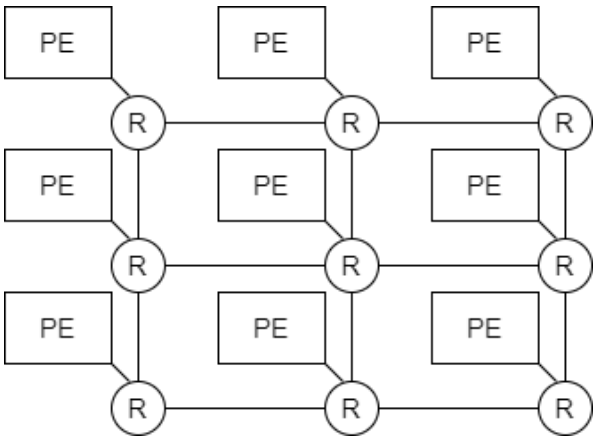


Figure 2.9: Basic NoC structure

While NoC is scalable to a large number of nodes in a communication system, it consumes a lot of power from its routers. The routing algorithm for routers could also be complicated to avoid contention and deadlock. The complex control algorithms are undesirable for asynchronous design since this will make timing checks more complicated. There is some research implementing NoC interconnect for SNNs. [15] proposed an NoC structure with a ring Router for SNNs. It achieves a low deviation of spikes' latency. But it is a synchronous design and does not mention the power data.

2.3.3 Segmented Bus

The segmented bus is a variant of the general bus topology. Its basic structure is shown in Figure 2.10. The segmented bus was first proposed to enable different nodes in the system to share common data, as well as sometimes communicate separately in small sub-blocks. [16] systematically builds a self-timed bus model and proposes a series of algorithms to optimize the total traffic in the system.

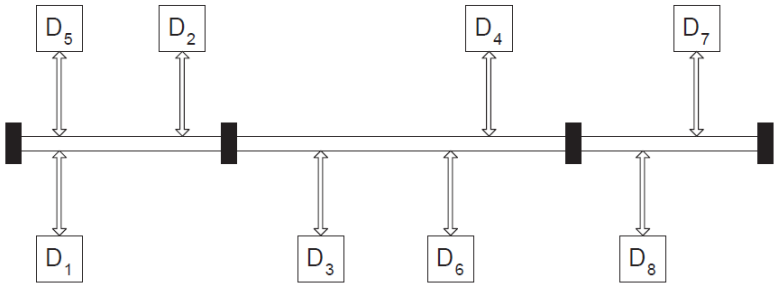


Figure 2.10: Segmented Bus topology[16]

Some interconnects use the segmented-bus topology to enable parallel communication and reduce power consumption. [17] proposes an asynchronous segmented-bus communication topology. It uses several handshake signals to arbitrate the use of different segments. [18] is a segmented-bus implementation SNN. It is a synchronous solution based on the software platform.

Point-to-point AER interconnect design

3

From the above-discussed background in Chapter 2, we find that a self-timed circuit is the best scheme to design the AER interconnect. Firstly, it is power-efficient compared with synchronous circuits as it does not have a global clock signal. On the other hand, compared with solutions that use handshake protocol, it has fewer control signals and less complex control schemes, thus leading to less area overhead and potential improvements in throughput. The most important point is that the self-timed circuit is greatly compatible with the working principle of SNNs, i.e., using spikes (events) themselves to contain timing information that they need to transmit.

This chapter designs an AER interconnect for point-to-point communication. It is organized as follows. In section 3.1, design considerations for the interconnect are put forward. Then in section 3.2, the whole structure is presented, and the details of how the interconnect works are also explained. Next, in section 3.3, the focus component, the priority encoder, is designed. Several priority encoders introduced in other literature are discussed, and a new kind of priority encoder is proposed and designed. Having finished the functional design, in section 3.4, timing problems in this self-timed design are analyzed, and several solutions are offered. Finally, section 3.5 summarizes this chapter.

3.1 Design Consideration

Figure 3.1 shows a highly abstracted point-to-point communication diagram. The transmitter encodes the spikes from NA1 into AER signals and sends them to NA2. In the point-to-point design, the transmitter is the critical part mainly because of 2 reasons.

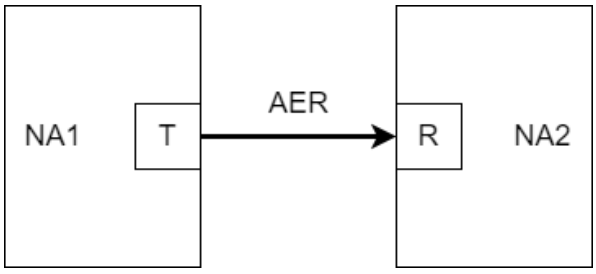


Figure 3.1: Highly abstracted diagram for point-to-point structure with T representing transmitter, R representing Receiver

Firstly, in NAs, different neurons may spike concurrently. So the transmitter should be able to deal with the simultaneous spikes and encode them into an organized AER sequence at the output.

Another reason is that if the transmitter is well-designed and meets all the timing requirements, the receiver in another NA can use simple combinational logic to decode the AER signal into corresponding spikes.

3.2 Whole structure

Figure 3.2 shows the whole structure of my designed transmitter. It mainly consists of 4 different parts, a sampler, a priority encoder, an address-handling unit, and a feedback loop.

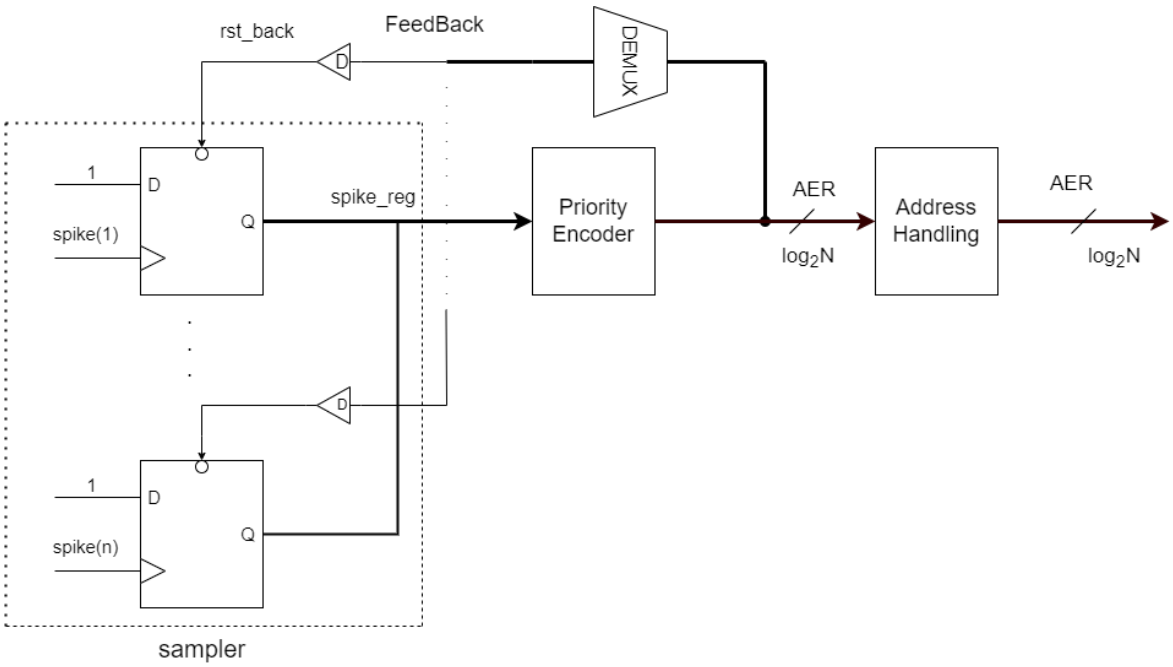


Figure 3.2: Structure of the proposed transmitter in this thesis

The sampler is composed of a set of flip-flops. Each flip-flop's clock port connects to an individual neuron to receive its spikes. The data port of the flip-flop is consistently set to logic 1. Each time the rising edge of a new incoming spike arrives at the clock port, the corresponding flip-flop's output is set to 1. Then after some delays, the output is reset to 0 again, indicating this spike has been successfully handled by the transmitter.

There are 2 options to build the sampler, which are using flip-flops or latches. Flip-flops detect the edge of the spikes, while latches detect the spike level signals. Since the neuron's spike width is usually in the unit of μs or ms[19], which is much higher than delays in modern digital circuits, it is a better option to choose the flip-flop to build the sampler. On the other hand, using latches could cause a re-sampling problem because the same spike could set the latch's output to 1 again, even if it is reset by the feedback loop.

The priority encoder is the core component of the transmitter because it needs to deal with simultaneous spikes from the sampler and output a final address corresponding to the highest priority.

The address-handling unit performs some simple operations on the AER signals from the priority encoder. It plus 1 to the AER signal since 0 is rendered invalid in our design. In multi-array communications, it also needs to append the index of source NA.

A feedback loop exists between the AER output and the sampler's reset ports. In this feedback loop, the DEMUX first decodes the final address to n-bit lines, and each bit line connects back to the reset port of the corresponding flip-flop through a delay element.

The latency of the transmitter is determined by equation 3.1. In this equation, T_{trans} is the cycle for the transmitter to sample and send one spike from the input. t_{pri_enc}, t_{demux}, t_{delay}, t_{rst2q} represents the delay of the priority encoder, DEMUX, delay element, and flip-flop's reset-to-q path respectively.

$$T_{trans} = t_{pri\_enc} + t_{demux} + t_{delay} + t_{rst2q} \tag{3.1}$$

The receiver has a much simpler structure than the transmitter designed in this thesis. Figure 3.3 (a) shows a diagram of the receiver. The decoder decodes the AER signal into spikes sequentially at the first stage. Then it is optional to use a spike stretcher to restore the spike width for neuronal communication inside the NA. The spike stretcher uses an SR latch to stretch the pulse width, as shown in Figure 3.3 (b). The final spike width is equal to the delay value of the delay element (expressed as a triangle with the letter D in it).

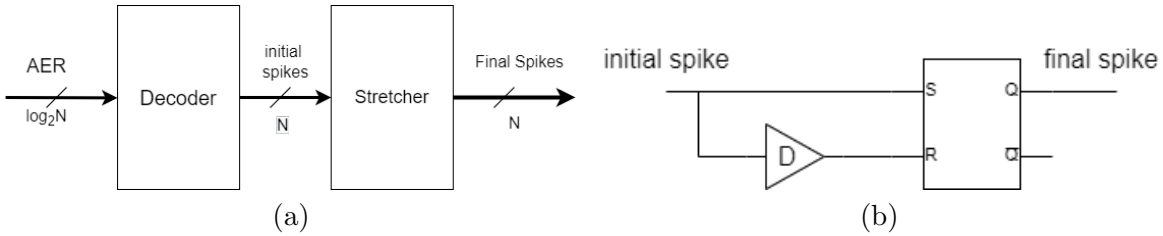


Figure 3.3: The diagram of (a) Receiver, and (b) Spike Stretcher

3.3 Priority Encoder Design

The priority encoder is the most important building block in the transmitter. In this section, different kinds of priority encoders are discussed and two kinds of priority encoders suitable for our project are proposed.

3.3.1 Commonly-used priority encoders

A priority encoder is a kind of circuit that can accept multiple valid bits and output a final address corresponding to the bit with the highest priority.

Generally speaking, there are 2 kinds of schemes to implement a priority encoder. The first scheme divides the priority encoder into a prioritizer and a one-hot encoder. The prioritizer takes N input bits and generates N output bits. There is only one valid bit at the output regardless of the number of valid bits from the input. Then the output from the prioritizer comes into a one-hot encoder where the final address is produced. The second scheme is to take the priority encoder as a whole. In this kind of scheme, the priority encoder takes N-bit inputs and directly generates a log_2N-bit address.

Figure 3.4 (a) shows an 8-bit priority encoder built with a prioritizer and an encoder. Using an 8-bit prioritizer(PRI) as a building block, Figure 3.4 (b) constructs a 64-bit priority encoder with a serial prioritizer. In this structure, the PRI_i needs to wait for the enable signal from PRI_{i-1} to function correctly. Therefore, the serial prioritizer has a quite long critical path. Figure 3.4 (C) exhibits a 64-bit priority encoder with a parallel prioritizer. This structure was first proposed by [20]. The prioritizer uses some OR gates and an 8-bit prioritizer to generate the enable signal for $PRI_0 \sim PRI_7$ in parallel. Then all the prioritizers in the second stage operate concurrently to generate a one-hot output. Although this structure has lower latency than the serial one, it adds some extra area overhead.

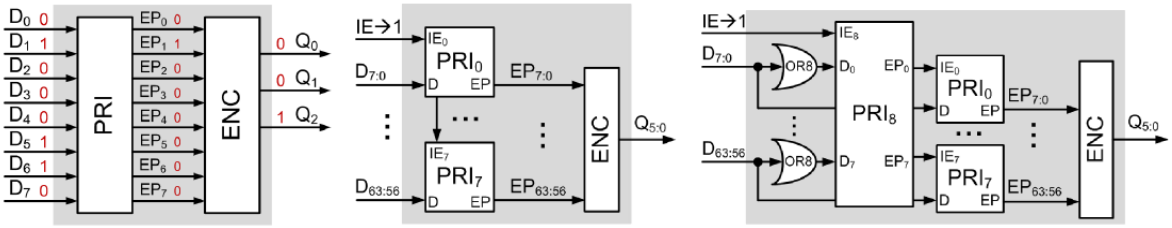


Figure 3.4: The block diagram of (a) PE8 composed of a prioritizer and an encoder, (b) 64-bit Priority Encoder with a serial prioritizer, (c) 64-bit Priority Encoder with a parallel prioritizer [21]

To improve the performance of the priority encoder, the author in [21] proposes a 1D-to-2D conversion scheme of the priority encoder. This approach considers an N-bit input as an $R \times C$ array. Taking 64-bit input as an example, the input bits are firstly converted to an 8×8 array, as shown in Figure 3.5. Each bit has a row index and a column index. The final address is the combination of the two indexes.

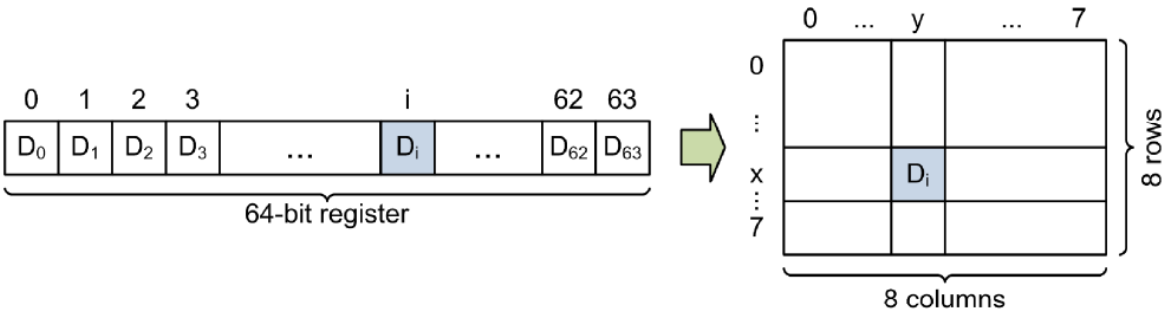


Figure 3.5: The conversion from 64-bit to 8x8 bit [21]

The implementation of a priority encoder utilizing 1D-2D structure is shown in Figure 3.6. The circuit works as follows. Firstly, the 64-bit input is divided into 8 8-bits groups. Each adjacent 8 bits come to an OR gate to generate a DOR signal which indicates if there exists at least one valid bit in this group. Then the 8-bit DOR signal enters an 8-bit priority encoder to generate the higher 3 bits. The higher 3 bits also functions as the select signal to decide which row could enter the second 8-bit priority encoder. Then the second priority encoder generates the lower 3 bits of the final address. The Final address is just a concatenation of 2 priority encoders' output.

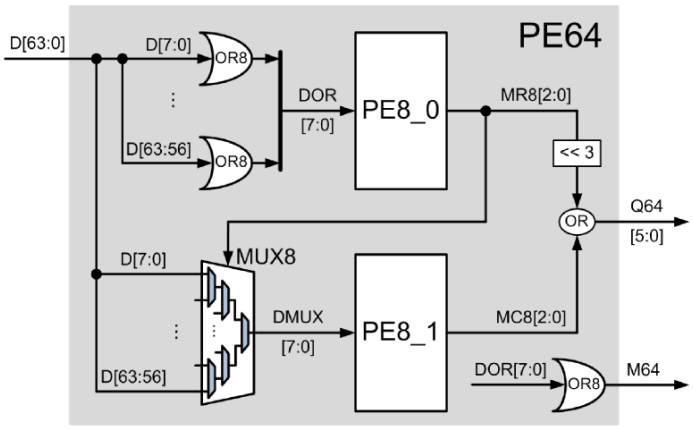


Figure 3.6: Implementation of an 1D-2D conversion priority encoder [21]

Based on the 1D-2D conversion structure, the author in [22] proposes 2 improvements. Firstly, the most suitable values for the number of columns and rows are found. The author points out that when the number of columns is 4, the priority encoder performs the best. Besides, a look-ahead signal is also added to decrease the propagation delay of the priority encoder. [23] presents their results of a 256-bit priority encoder after fabrication.

The priority encoder with 1D-2D conversion greatly saves the area. However, it has a complicated wire routing scheme. Therefore, it is unsuitable for designs that may encounter wire delay and congestion problems.

Some other articles also put forward different kinds of measures to improve the performance of the priority encoder. [24] suggests a 2D-array to 3D- array conversion method for priority encoder. This idea is similar to 1D-2D conversion, and it further extends the dimension of the input bits. But it also increases the wiring complexity. Some articles try to enhance the priority encoder at the transistor level. [25] and [26] do a full-custom design and generate a regular layout that improves the timing, area, and congestion. A full-custom design often outperforms a similar semi-custom design, but it is impracticable to large digital systems.

3.3.2 Tree-structure priority encoder with rotated priority

The priority encoder in this project is used to transmit spikes, and it connects directly with NAs. Therefore, we need to consider some specific features and constraints of SNNs.

Firstly, Neuron circuits are much larger in size compared with priority encoders. [27] designs an SNN chip with 384 neurons and 98304 synapses embedded. The chip has a $5 \times 5\ mm^2$ core size, and the synapse array occupies most of the area of the chip. In contrast, in [23], a 256-bit priority encoder only occupies an area of $0.78mm^2$. The disparity between neuron array and priority encoder brings about 2 consequences. Firstly the total wire length becomes very large, and the wire delay may dominate the circuit delay. Besides, the length of different wires in the priority encoder may cause a high delay dispersion. Therefore, we need a highly regular structure to minimize wire delay and dispersion.

Secondly, AER signals appear in sequence at the output of the priority encoder. This may cause some delay(timing errors) for simultaneous spikes. We need to average this timing error among different neurons.

Taking these 2 aspects into account, two tree-structured priority encoder with rotated priority is proposed.

3.3.2.1 The separate prioritizer and one-hot encoder solution

The first solution is a structure with a separate prioritizer and a one-hot encoder. Figure 3.7 shows an 8-bit tree prioritizer built with Building Blocks (marked as BB). At every stage of the tree prioritizer, each pair of adjacent outputs from the previous stage serves as the input for the next stage's Building Block. An N-bit input needs to pass through $log_2 N$ stages to generate a valid output. The property of priority rotation is embedded into each Building Block.

Figure 3.8 exhibits the structure of the Building Block. It takes a pair of n-bit signals, together with their enable signals, as input. The enable signal signifies if there is at least one valid bit in the input. For the first stage of the prioritizer where n is equal to 1, the IE_1 and P1 port both connect to the same bit of the sampler's output, as well as IE_2 and P2. The Building Block generates a 2N-bit output as well as an enable signal and sends them to the next stage. Table 3.1 describes the function of the Building Block.

The flip-flop in the Building Block plays the role of inverting the priority. When the input enables signal switches from 11 to 10,01 or 00 (actually this does not happen

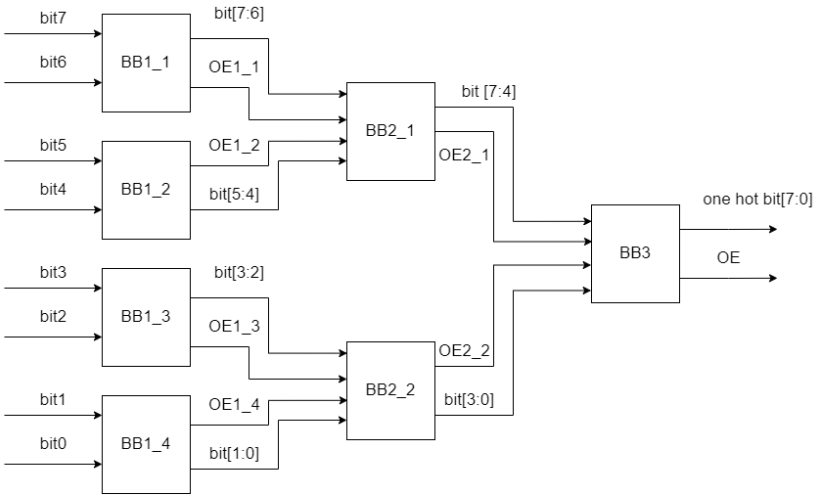


Figure 3.7: Proposed 8-bit prioritizer.

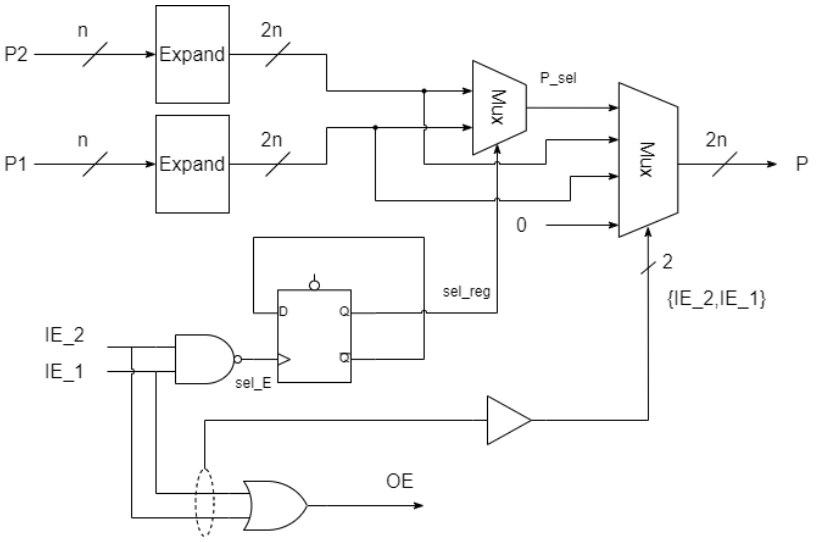


Figure 3.8: Diagram of Building Block for prioritizer.

because each time only one bit can be encoded), the priority of the Building Block toggles. The falling edge of the enable signals is used to trigger the flip-flop in order to avoid contention in different paths.

The one-hot encoder can accept the prioritizer's output and generate a final AER signal. Since there is only one valid bit for the encoder's input, it is also possible to use a tree encoder to decrease the wire length, as shown in Figure 3.9.

With a separate prioritizer and encoder structure, the previous transmitter structure in Figure 3.2 also needs to be altered a little bit. The changed transmitter structure is shown in Figure 3.10.

Table 3.1: The function of Building Block in prioritizer

IE_2	IE_1	sel_reg	P	OE
0	0	x	0	0
0	1	x	{n{0},P1}	1
1	0	x	{P2,n{0}}	1
1	1	0	{n{0},P1}	1
1	1	1	{P2,n{0}}	1

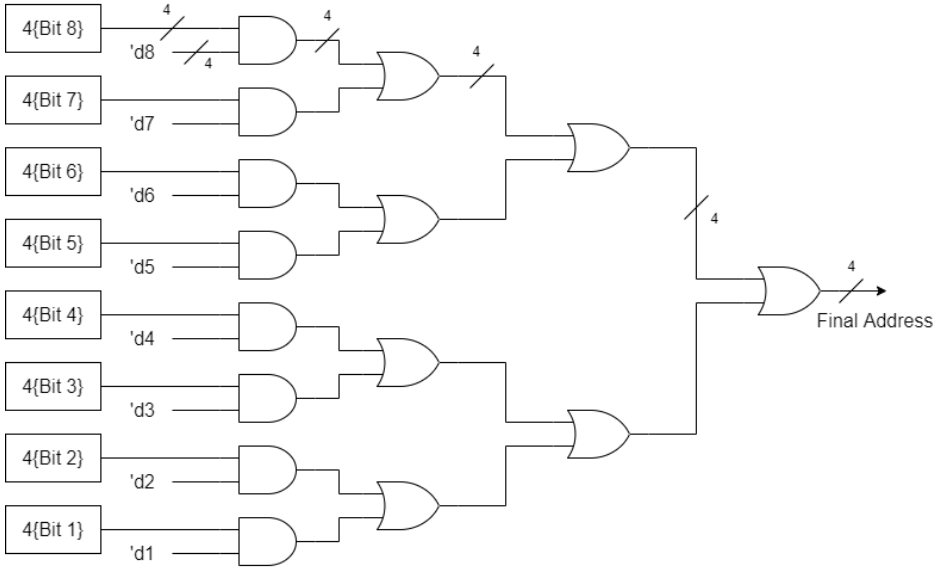


Figure 3.9: Structure of a one-hot tree encoder with 8-bit input.

Similar to the previous structure, the latency of the transmitter is determined by equation 3.2. t_{pri}, t_{inv}, t_{delay}, t_{rst2q} represents the delay value of the prioritizer, inverter, delay element, and flip-flop's reset-to-q path respectively. Equation 3.3 is another constraint on this structure. t_{enc} is the propagation delay of the one-hot encoder. This equation simply means that the one-hot encoder should finish the encoding process within a transmission cycle.

$$T_{trans} = t_{pri} + t_{inv} + t_{delay} + t_{rst2q} \tag{3.2}$$

$$T_{enc} \leq t_{pri} + t_{inv} + t_{delay} + t_{rst2q} \tag{3.3}$$

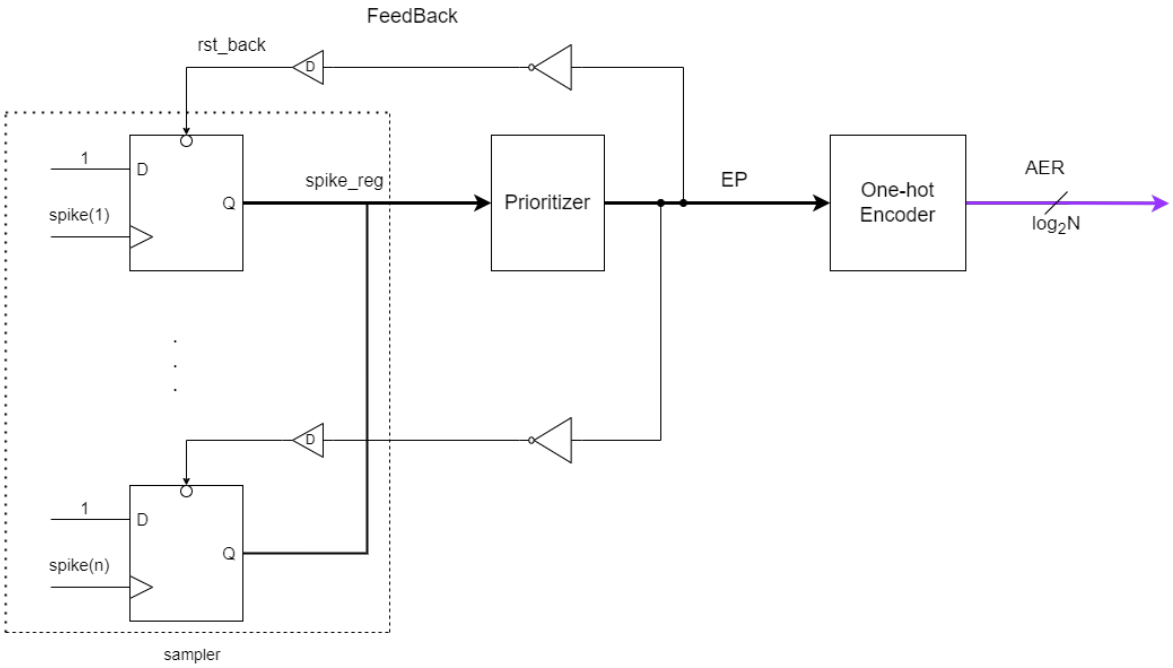


Figure 3.10: Structure of transmitter with separated prioritizer and one-hot encoder.

3.3.2.2 The priority encoder solution

If we observe the binary coding from 0 to N, the lowest bit is different for each pair of adjacent numbers. Then the second rightmost bits are the same for numbers 2n and 2n+1. The third bits are the same for continuous numbers from 4n to 4n+3. This coding property enables us to implement a tree priority encoder, as shown in Figure 3.11. It has a similar structure to the tree prioritizer. Each stage in the tree priority encoder encodes 1 bit of the address. So in this structure, the final address is determined bit by bit. It is worth noting that there is a plus 1 block at the end of the priority encoder. This is because 0 is an invalid signal in AER communication and is therefore deprecated.

Figure 3.12 shows the structure of the Building Block of the proposed encoder. It is very similar to the Building Block for prioritizer. The only difference is that it takes in an n-bit temporary address and outputs an n+1-bit temporary address to the next stage. The function of this Building Block is illustrated in 3.2.

Table 3.2: The function of Building Block in prioritizer

IE_2	IE_1	sel_reg	P	OE
0	0	x	0	0
0	1	x	{0,P1}	1
1	0	x	{1,P2}	1
1	1	0	{0,P1}	1
1	1	1	{1,P2}	1

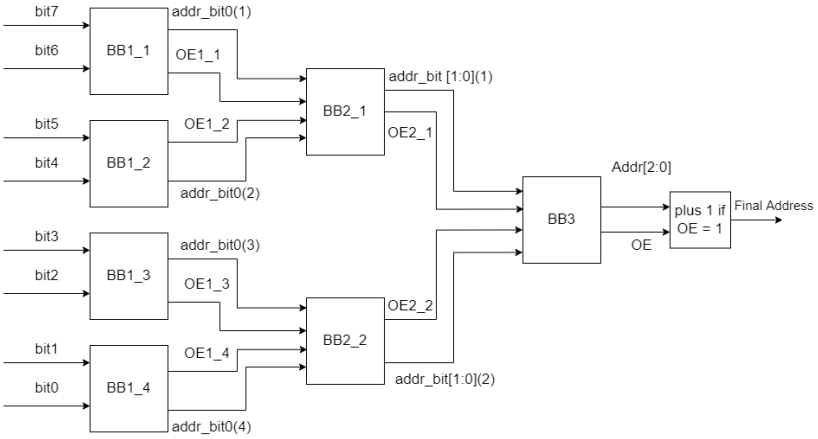


Figure 3.11: Structure of tree priority encoder.

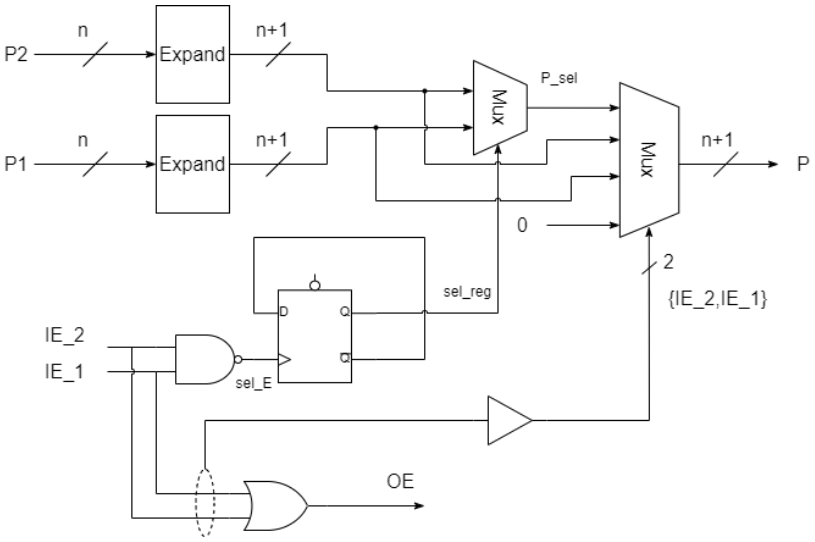


Figure 3.12: Structure of the Building Block for priority encoder.

3.4 Solving timing problems

Solving timing problems is the most significant part of the self-timed design. There are mainly two kinds of timing problems introduced by the transmitter. In the following subsections, these problems will be tackled respectively.

3.4.1 Essential Hazards

Essential hazards are caused by the difference in the propagation delay in two or more paths. When it happens, a short glitch or unstable state may appear at the output. The glitches caused by essential hazards in the transmitter are shown in Figure 3.13. In synchronous circuits, this kind of hazard can be easily resolved if the circuit meets the setup and hold timing requirements. However, in self-timed circuits, due to the lack of a clock signal, the glitches may propagate and accumulate throughout the whole

circuit.

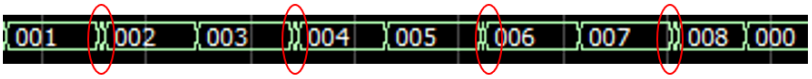


Figure 3.13: Glitches caused by essential hazards in the transmitter (in red circles).

Figure 3.14 shows the essential hazard in the prioritizer's Building Block. When two input enable signals $\{IE_2, IE_1\}$ change from 11 to 10 or 01, three paths may propagate to output at different times:

- Path1: From input enable signals, through the flip-flop and 2 MUX, to the output (blue line). This path has a delay of $t_{nand} + t_{pff} + t_{m1} + t_{m2}$. However, this is a false path because the output of the flip-flop does not play a role when $\{IE_2, IE_1\}$ = 01 or 10.
- Path2: From input enable signals, through the second MUX, to the output (green line). This path has a delay of t_{m2}.
- Path3: From P1 or P2, through the expand block and 2 MUX, to the output. This path has a delay of $t_1 + t_{pe} + t_{m1} + t_{m2}$ (assume P1, P2 arrives t_1 later than IE_1 and IE_2.

Due to the different delays in Path2 and Path3, a glitch of $t_1 + t_{pe} + t_{m1}$ width may appear at the output of the Building Block.

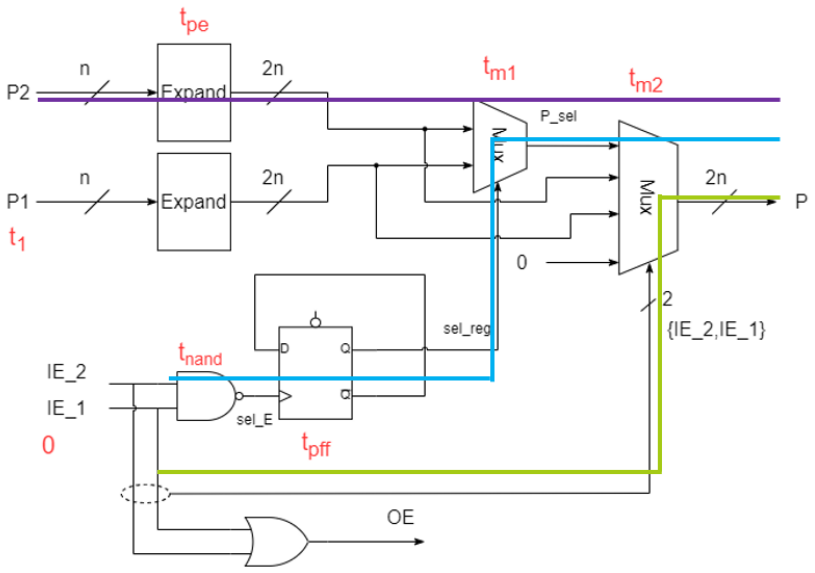


Figure 3.14: Essential hazards in prioritizer's Building Block. The delay of each element is colored red and the different paths are indicated in different colors

To solve the timing problem caused by essential hazards, three methods are proposed.

The first method is to do the delay matching, as Figure 3.15 shows. This method adds a delay element in the shorter path. The delay value of the delay element is constrained by equation 3.4. t_{delay} and t_{next_min} are the delay values of the delay element and the minimum pulse width that could be handled by the next stage. Other terms are different elements' delays indicated in red in Figure 3.14. The equation indicates that after delay matching, the difference in propagation delay in 2 paths should be smaller than the minimum pulse or data width that the next stage can deal with so that the glitch can not propagate and accumulate through the circuit.

$$|(t_{delay} + t_{m2}) - (t_1 + t_{pe} + t_{m1} + t_{m2})| \leq t_{next\_min} \tag{3.4}$$

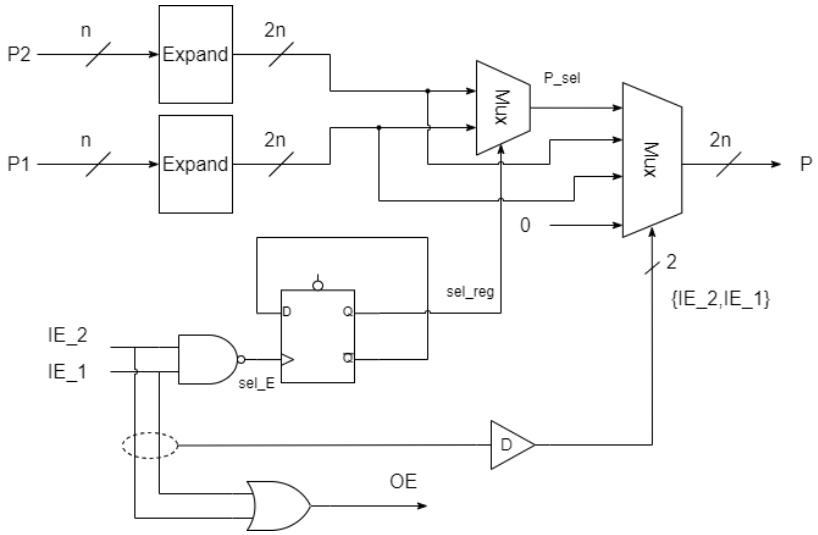


Figure 3.15: Delay matching method to solve the essential hazards

The second approach also adds delay elements to the circuits, but the principle behind them differs. Instead of doing the delay matching, this method uses the inertial delay of the delay element to filter out unwanted glitches. Inertial delay is a built-in property of circuit elements. [28] points out that it is not possible to build a device with an arbitrarily small decision window. In other words, a circuit element cannot handle signals with a randomly small duration time. Due to the parasitic capacitance and resistance, it will filter out signals that have a less duration time than its inertial delay. Then based on this theory, [29] builds a more accurate inertial and degradation model for CMOS logic gates.

Figure 3.16 shows the method which adds a delay element at the output of the circuit to filter unwanted glitches. The delay element is constrained by equation 3.5. Here t_{del_iner} represents the inertial delay of the delay element. The rest items have the same meaning as in equation 3.4.

$$t_{del\_iner} \geq t_1 + t_{pe} + t_{m1} \tag{3.5}$$

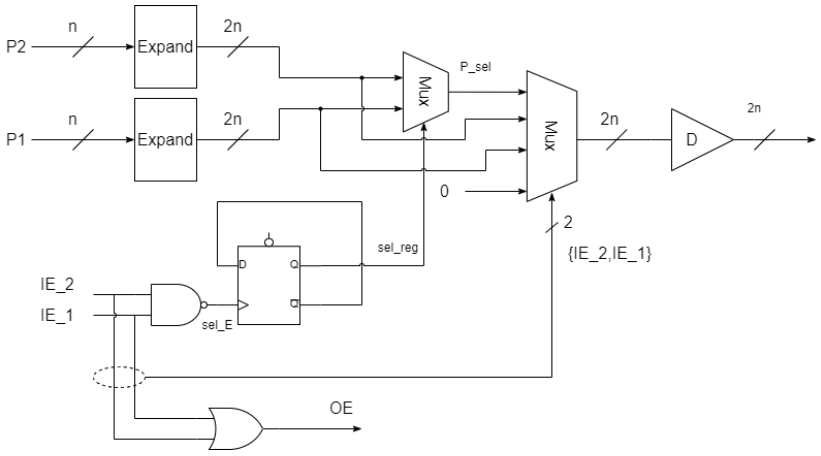


Figure 3.16: Using inertial delay to filter unwanted glitches

The third solution uses a series of Asynchronous Finite State machines (FSM) and an event-driven flip-flop to solve the glitches in the transmitter.

This solution is based on the fact that in each transmission cycle, one bit from the sampler's output is reset to 0. The falling edge of this reset operation indicates that a new bit is sampled by the prioritizer or handled by the priority encoder. Thus it is possible to sense these falling edges by an asynchronous FSM.

Asynchronous FSM is a kind of circuit element that can memorize and change the states in asynchronous circuits. Figure 3.17 shows a basic diagram of Asynchronous FSM. Similar to synchronous FSM, it also uses the present state to determine the next state of the circuit and form the output. However, it differs from the synchronous FSM in that instead of using a global clock signal to memorize the state and form the next state, it uses delay elements to push the state transition. The asynchronous FSM will be discussed in detail in Chapter 4.

Figure 3.18 shows the state transition diagram of our proposed asynchronous FSM. There are four states in total, three of which are idle states. The only valid state is the sample state. The spike signal in the diagram represents the sampled spike, i.e., one bit of the sampler's output. When this bit transfers from 1 to 0, the state changes from Idle1 to Sample, triggering a short pulse, and this pulse is used to drive the event-driven flip-flop.

Figure 3.19 shows the structure of the transmitter(separate prioritizer + one-hot encoder solution) with the proposed FSM + event-driven flip-flop solution. The logic OR of all the FSM's output is the final pulse to drive the event-driven flip-flop. The address signal from the one-hot encoder also passes through an OR gate and connects to the reset port of the flip-flop.

There are 2 timing constraints for the event-driven flip-flop. Inequality 3.6 means that the setup requirements for the event-driven flip-flop must be met. t_{hop} is the delay value of the delay element in asynchronous FSM, which is also the time for a single hop from one state to another. t_{p2ck} represents the path delay from the outputs of FSMs to the flip-flop's clock port. And t_{setup} is the setup time for the flip-flop. Inequality 3.7 ensures that the flip-flop can sample the data in the current transmission cycle in case

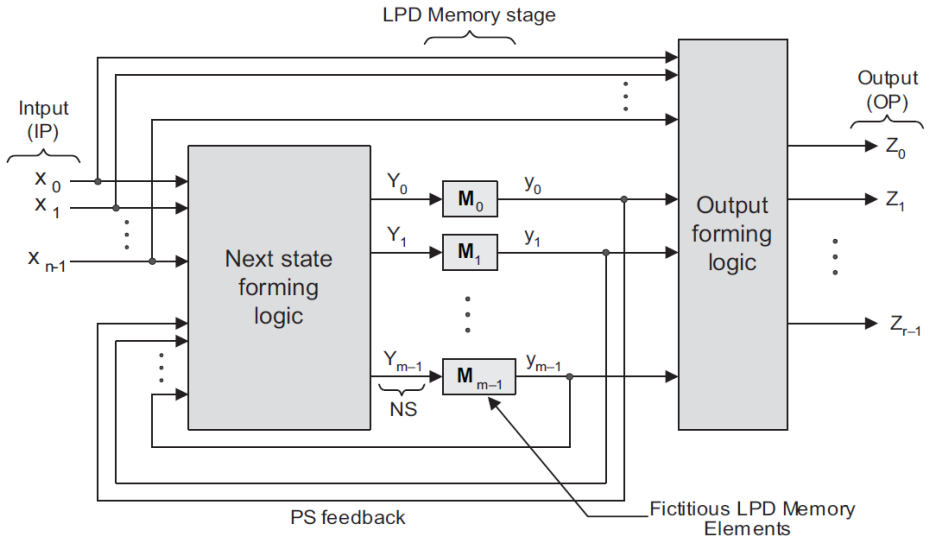


Figure 3.17: The diagram for Asynchronous FSM[30]

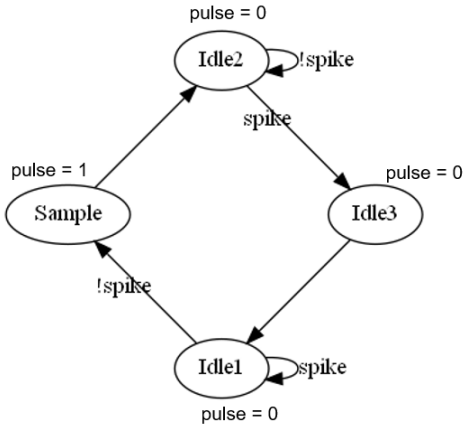


Figure 3.18: The state transition diagram of the proposed FSM

the data is lost.

$$t_{inv} + t_{delay} + t_{rst2q} + t_{hop} + t_{p2ck} \geq t_{enc} + t_{setup} \tag{3.6}$$

$$t_{hop} + t_{p2ck} < t_{pri} + t_{enc} \tag{3.7}$$

As for the whole priority-encoder structure, the corresponding timing constraints are shown in inequality 3.8 and equation 3.9. The t_{han} in the inequality represents the delay of the address handling circuit.

$$t_{demux} + t_{delay} + t_{rst2q} + t_{hop} + t_{p2ck} \geq t_{han} + t_{setup} \tag{3.8}$$

$$t_{hop} + t_{p2ck} < t_{enc} + t_{han} \tag{3.9}$$

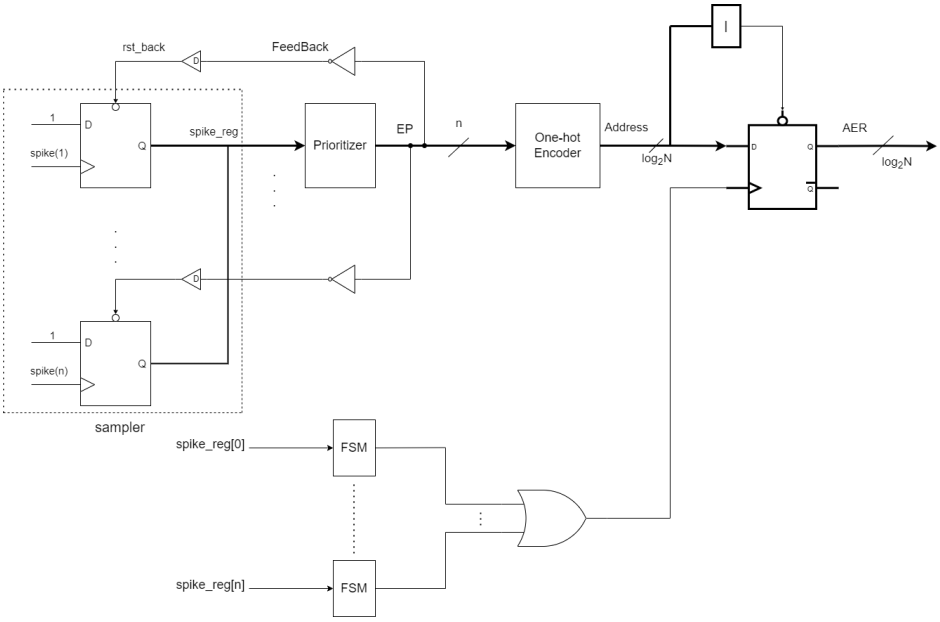


Figure 3.19: Transmitter with event-driven flip-flop

Taking all these 3 solutions into account, a brief comparison of them is given in Table 3.3 in terms of added overhead and how strict the timing constraints are. The first delay matching solution has a high extra overhead and the strictest timing requirement. The second solution tends to have the lowest added overhead (not definitely, but it also depends on the specific implementation of the delay element). And its timing requirement for added delay elements is very loose. In contrast, solution 3 also has a high area overhead because it needs a set of asynchronous FSM, a $log_2 n$-bit event-driven flip-flop, and possibly some delay elements. Its timing constraints are also relatively loose. On the other hand, solution 3 does not break the structure of the whole priority encoder, while for the first and second solutions, the delay element needs to be added at each Building Block inside the whole priority encoder. As a result, the priority encoder cannot be synthesized as a whole.

Table 3.3: A comparison between 3 glitch-solving solutions

Solution	Overhead	Timing requirement
1	2(n-1) delay element	strict
2	n($log_2 n$) delay element	very loose
3	n FSM + $log_2 n$ FF + (n delay element)	loose

3.4.2 Glitches caused by close incoming spikes

Another kind of glitch may also occur due to the very close input spikes. It happens when a spike comes closely after a previous spike, but not simultaneously. This may cause the Building Block of the priority encoder to change its value in an extremely short period. This kind of glitch is not the wrong data. It just lasts shorter than a

normal transmission cycle. However, it can also lead to the malfunction of the circuit in some cases. In the rest of this subsection, this kind of glitch is discussed based on the 2 transmitter structure proposed earlier in this chapter.

As for the transmitter described in Figure 3.10, a glitch may occur at the output of the prioritizer with the highest possible width of t_{pri}. Three possible scenarios exist for the glitch as follows.

- If the glitch passes both the one-hot encoder and feedback loop, the circuit functions correctly. The same applies if the glitch passes neither the encoder nor the feedback loop.
- If the glitch is processed by the one-hot encoder but not fed back to the reset port of the sampler, the glitch will also appear at the output of the encoder. This can be solved in 2 ways. The first one is to add delay elements at the output of the encoder, with an inertial delay greater than t_{pri}. The event-driven flip-flop mentioned previously could also resolve glitches in this situation.
- If the glitch feeds back to the reset ports of the sampler but is not processed by the encoder, a spike is lost. To prevent this from happening, the inertial delay of the feedback's delay elements should be larger than the encoder's propagation delay, as expressed in equation 3.10.

$$T_{delay\_iner} \geq t_{enc} \tag{3.10}$$

Similar cases apply to the transmitter in Figure 3.2. Here the constraint is expressed in 3.11, which means that the inertial delay of the feedback loop's delay element should be greater than the propagation delay of the address handling circuit. In other words, a glitch that can be processed by its successor circuit is not a "glitch" anymore.

$$T_{delay\_iner} \geq t_{han} \tag{3.11}$$

Based on the above discussions, the event-driven flip-flop together with a requirement for the delay element's inertial delay is the best choice to resolve glitches in the transmitter. The event-driven flip-flop can resolve the glitches on the forward path of the transmitter and it does not affect the transmission cycle. The inertial delay adds a pulse-filtering feature to each delay element in the feedback loop. The pulses with a shorter width than the inertial delay will not be fed back to the sampler.

3.5 Summary

This section mainly contributes the following aspects:

- Build the whole structure of the transmitter of NA.
- Propose and implement 2 structures of the key element, i.e., priority encoder in the transmitter.
- Analyze and solve the timing problems in the transmitter.

Segmented Bus structure for multi-array communication

4

This chapter proposes a segmented bus topology for multi-array communication. Firstly, the motivation for the proposed topology is introduced. Then the topology is explained thoroughly, from the broad structure to the detailed elements design.

4.1 Motivation

When it comes to multi-array communication systems, some extra aspects need to be considered for the design. They are listed as follows:

- Firstly, a forward path and a backward path (corresponding to recurrent network) may both exist in SNN. So the communication system should be able to realize the function of bi-directional data transmission.
- Next, spikes from different NAs may arrive at another NA simultaneously. To avoid data loss, one of the data sources needs to stop sending data to allow the other source's data to be processed first.
- Lastly, sometimes an NA may only communicate with its adjacent NA. While on other occasions, it may communicate with several other NAs concurrently. This requires that the system should be capable of having its different sub-parts communicate simultaneously, as well as sharing data among multiple sub-parts or even the entire system.

Based on the above considerations, a self-timed segmented bus topology for multi-array communication is proposed and implemented.

4.2 The whole structure

Figure 4.1 shows the architecture of the proposed segmented bus for multi-array communication. In this figure, we design and implement a segmented bus structure that consists of 4 NAs, 1 AER input, 1 AER output, and 5 Fences. The terminology "Fence" is a new circuit element we design to control the operation of the multi-array interconnects. The structure and composition of the Fence will be discussed in detail in the next section.

Both a forward path and a backward path exist in this interconnect. The forward path is represented by a line with a right arrow, and the backward path is represented by a line with a left arrow. The NAs here are some spike generators that can generate different traffic patterns. As for AER signals, source encoding is used, with the highest 3 bits representing the array index (indicated by the 3-bit number inside the NA in Figure 4.1) and the rest bits representing the intra-array address.

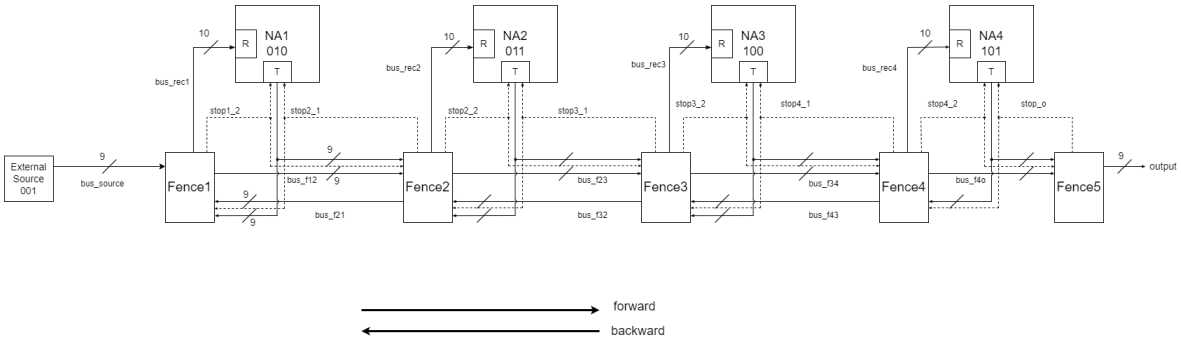


Figure 4.1: Self-timed segmented-bus architecture for multi-array communication

To illustrate the connections between NAs and Fences, NA_n and $Fence_n$ are used to represent the n-th NA and Fence starting from the left. The NA_n's transmitter can send AER signals to $Fence_{n+1}$ on the forward path and $Fence_n$ on the backward path. For the forward path, the NA_n's output AER signal, together with the AER signal from $Fence_n$, forms $Fence_{n+1}$'s forward input. A similar connection applies to the backward path of NA_n and $Fence_n$. To arrange the simultaneous AER inputs for the Fence, a stop signal (indicated in the dashed line in Figure 4.1) is needed to stop the adjacent NA's transmitting process. A Fence can send 2 stop signals, taking $Fence_n$ as an example, one to NA_{n-1} on the forward path and another to NA_n on the backward path.

The stop signals from adjacent Fences are cross-coupled to avoid deadlock. The scenario and waveform of deadlock in the system are shown in Figure 4.2 and Figure 4.3. Without the help of stop signal cross-coupling, when signal stop3_1 and stop1_2 become 1 simultaneously at some time, a closed loop forms among $Fence_1$, $Fence_2$, $Fence_3$ where all the transmission in this loop halts. What's worse is that this closed loop forms positive feedback in which stop3_1 and stop1_2 signal can boost each other to keep in the same state. As a result, this part of the communication system stops working as of the time when the deadlock happens, and the spikes from NA_1 and NA_2 cannot be encoded and transmitted to other NAs anymore. The cross-coupled stop signal can break the deadlock condition since Fences can accept other NA's stop signal and use it to prevent adjacent stop signals from staying in 1 continuously.

4.3 Fence Design

The Fence is crucial to control and ensure the communication system's regular operation. It has some main functions listed as follows.

- It should be able to connect and isolate different NAs at different times.
- It needs to control which AER signal can pass through the Fence to the next stage and which cannot.
- It should be able to stop the operation of the transmitter in NAs.
- It should be capable of sending AER signals to the receiver in its corresponding NA.

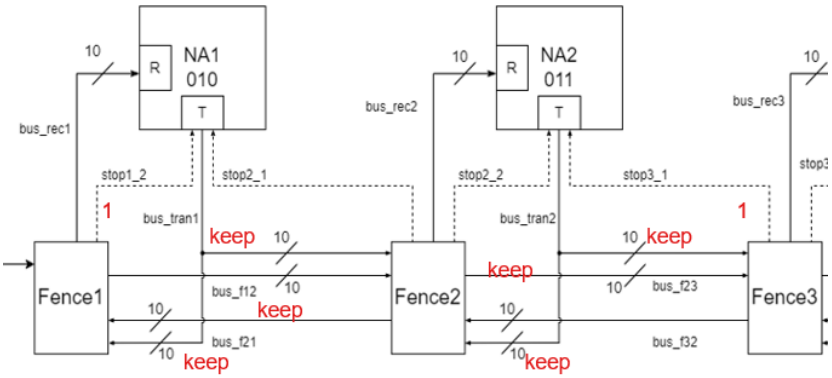


Figure 4.2: The diagram of the deadlock scenario. The stop signals here are not cross-coupled.

Figure 4.3: The waveform of the stop signals when deadlock happens.

Figure 4.4 shows the Fence structure. It has 2 symmetric forward and backward paths. Taking the forward path as an example, it has 2 AER inputs, one from the corresponding NA and another from the previous Fence. The 2 AER signals first come into 2 instances of the forward control logic unit (shown by "Forward TR Control" in rectangles). The forward control logic determines which AER signals can pass to the next stage and which cannot. The control logic can contain simple gates or a memory that specifies the destination of each AER signal. It is based on specific SNNs. Then the forward select logic can select and pass the AER signals from the control logic. Besides, it can also send a stop signal to the corresponding NA if needed. After that, the self-timed shift register (shown by "ShiftReg" in rectangles) ensures a fixed duration time for each AER signal. Apart from that, it also needs to send a control signal to the tri-state buffer. The tri-state buffer (indicated as a three-port triangle) can turn on or off to connect or isolate different Fences based on the control signal. Finally, AER signals from both forward and backward buses come to an asynchronous FSM to be sent to the corresponding NA's receiver in sequence. The backward path has a similar structure to the forward path except for the backward control logic because it is also determined by the specific SNN's composition and function. The select logic, FSM, and self-timed shift register will be discussed in detail in the following subsections.

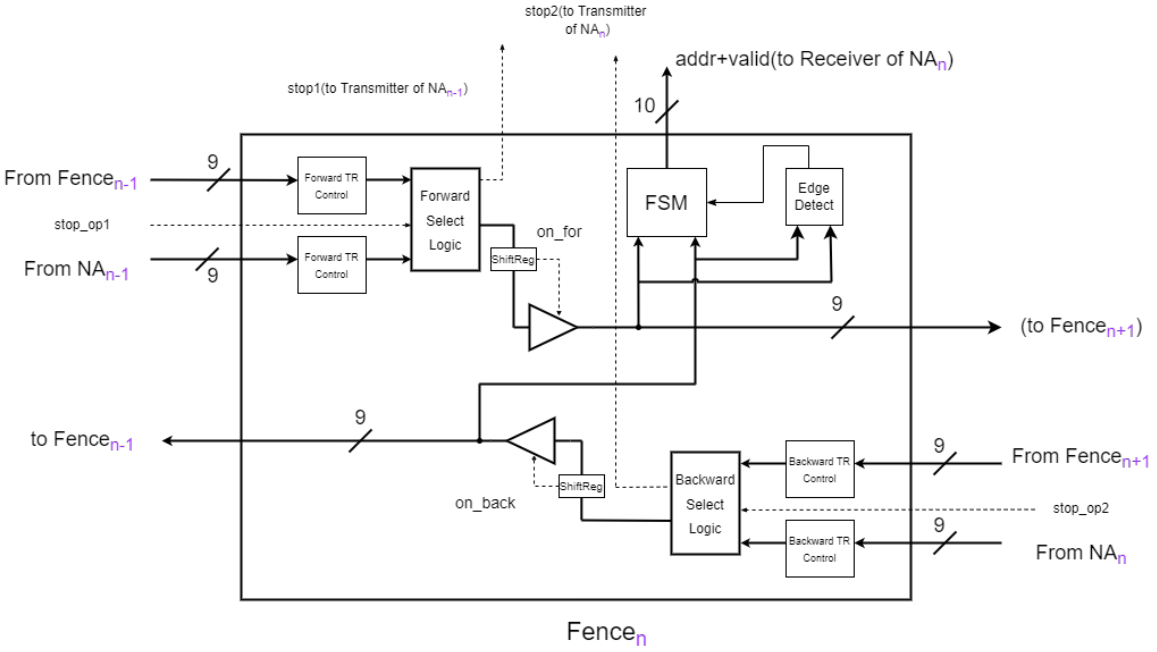


Figure 4.4: The structure of the Fence.

4.3.1 Select Logic

Table 4.1 shows the truth table of the Select Logic block in the Fence. There are two inputs, bus1 and bus2. Bus1 is the AER signal from the previous Fence, and bus2 represents the AER signal from the corresponding NA's transmitter. Index1 and index2 are the highest 3 bits of bus1 and bus2, respectively. They represent which NA they come from and are a non-zero number for a valid AER signal. The first four rows of the table need to be discussed.

Table 4.1: Function(truth table) of the Select Logic

index1	index2	stop_in	bus_out	stop_out
!=0	!=0	x	bus1	1
!=0	0	x	bus1	0
0	!=0	1	0	0
0	!=0	0	bus2	0
0	0	x	0	0

When two input buses both carry a valid AER signal, the Select Logic will pass bus1's AER signal and send a stop signal to the NA from which bus2 comes. If bus1 has a valid AER signal, but bus2 does not, the Select Logic will pass that signal, and no stop signal is generated. On the other hand, if only bus2 carries a valid AER signal, the input stop signal (indicated by stop_in in the table) from the opposite Fence will play a role in determining Select Logic's behavior. If stop_in is 1, which means that the NA bus2 comes from has stopped transmitting data, the output of the Select Logic will be set to 0 to prevent the deadlock (discussed previously in Figure 4.2) from happening.

Otherwise, it will pass bus2 to the output.

The function of the stop signal is shown in Figure 4.5. The input stop signal can suspend the transmitting process by stopping the feedback loop. When the input signal is valid, the Mux will select an all-one value instead of the value from the prioritizer to send to the sampler's output ports. Therefore, the output of the prioritizer and encoder remains the same value, and the transmitter stops working.

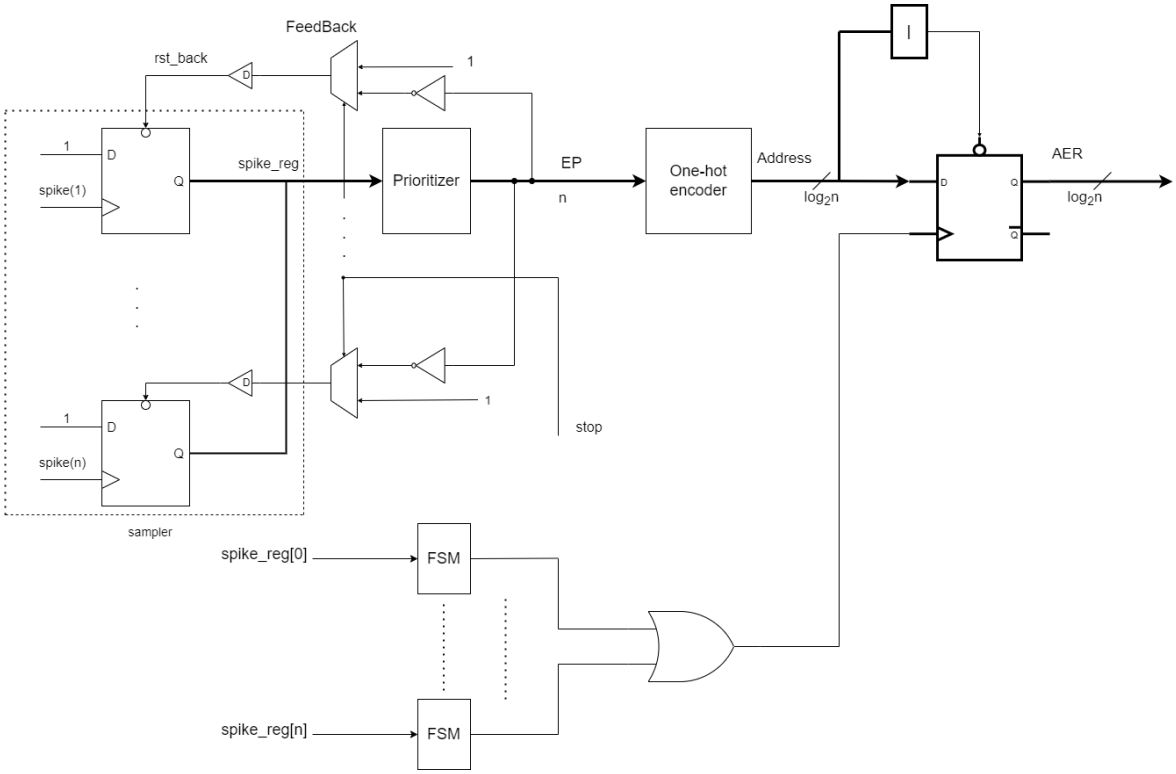


Figure 4.5: The transmitter with an input stop signal.

4.3.2 Asynchronous FSM

Asynchronous FSM in the Fence collects output AER signals from both forward and backward paths. The concept of asynchronous FSM has been introduced in Chapter 3. In this section, it will be discussed in more detail. Then the Asynchronous FSM in the Fence is designed.

According to [13], there are 2 kinds of asynchronous FSMs, the ones that operate in fundamental mode and the ones that work out of fundamental mode. In the fundamental mode of FSM operation, only one input of the FSM is allowed to change at a time. Another change at a different input is allowed to happen only after the FSM has stabilized. A stabilized FSM means that the next state is the same as the present state for current inputs. If the input does not meet the timing requirements stated above, the FSM should operate in non-fundamental mode.

The delay element differs in FSMs operating in fundamental mode and non-fundamental mode. In fundamental mode, any circuit elements with a certain delay

value can be used, such as SR latches or a chain of inverters. But in non-fundamental mode, there is less choice. This is because the normal delay elements, such as buffers and inverters, may fall into meta-stability due to the fast-changing inputs. The Muller-C element is a commonly used delay element for FSMs operating in non-fundamental mode. Figure 4.6 shows the circuit and truth table of the Muller-C element. It has two stages. The first stage comprises 4 transistors connected in series, and the second stage is a pair of keep inverters. When two inputs are both high or low, the output is set to 1 or 0, respectively. When 2 inputs differ, the circuit will maintain its current state. Because of the keep inverter, the Muller-C element can avoid the meta-stability problem.

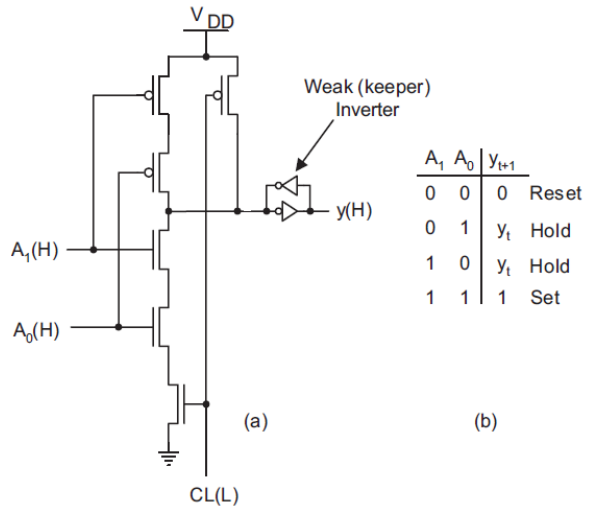


Figure 4.6: Muller C elements for FSMs operating in non-fundamental mode (a) circuit model (b) truth table.

The FSM in the Fence collaborates with an edge detector to send AER signals to the receiver in sequence. Figure 4.7 shows the diagram of the edge detector. When a new AER signal comes, it can generate a valid signal with a certain width by comparing the AER signal with its delayed value.

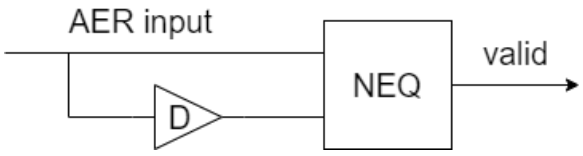


Figure 4.7: The diagram of Edge Detector.

The structure of the FSM that we use in the Fence is shown in Figure 4.8. The FSM operates in non-fundamental mode because both forward and backward buses can change simultaneously. The data from one bus may also change at any time before or after the other one. The 2 valid signals from the forward and backward bus are used together with the present state (indicated by y_0 and y_1) to determine the next state

of the FSM. Each bit of the next state from the Next State forming logic is converted into 2 bits to drive a Muller-C element. Then the present state is generated by a set of Muller-C elements after a certain delay. The output forming logic uses present state and AER signals from both the forward and backward bus to produce the final output that will be sent to the corresponding receiver.

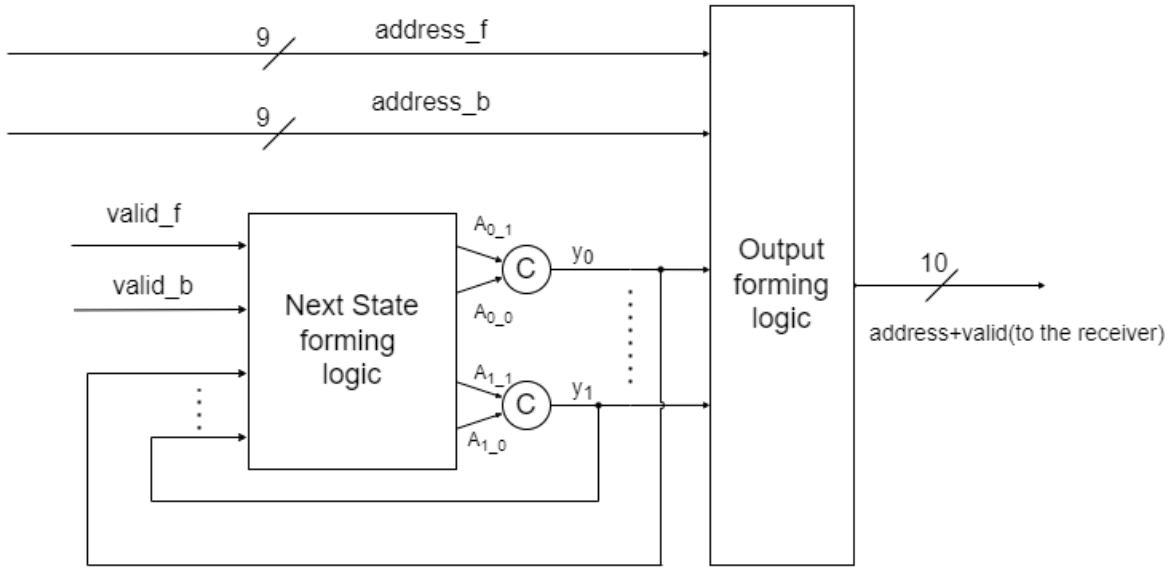


Figure 4.8: The FSM we designed for the Fence.

Since the AER signals from forward and backward buses can have any phase relationship, the FSM must be robust to deal with all situations. This requires the FSM to sample each AER signal from 2 buses and avoid the re-sampling or loss of data.

The state transition diagram of the FSM is shown in Figure 4.9. It contains three loops. The first loop (consisting of A and a set of B states) represents the situation where two valid signal changes simultaneously. In this case, the FSM will sample each AER input in sequence and then return to A state. The other 2 loops describe a similar scenario where one of the valid signals changes at first. Then another valid signal also changes after some time, with the first valid signal still in logic 1. These 2 cases have more branches than the first one.

To make the FSM operate correctly in different input conditions, some constraints need to be added. Assume that the width of logic 1 and logic 0 of valid signal in a transmission cycle is t_{v1} and t_{v0}, respectively. Then Equation 4.1 holds where t_{AER} is the duration time for an AER signal. The delay value of Muller-C elements in FSM is t_d. This value also indicates the time it takes to finish a state transition. Then three extreme cases for the timing constraints are illustrated in Figure 4.10. Valid1 and valid2 in the figure represent the valid signal for forward and backward AER signals. The first arrow's start point of each case, which locates in the middle of valid1 and valid2 signal, indicates state A shown in Figure 4.9. The direction of each arrow represents the state transition.

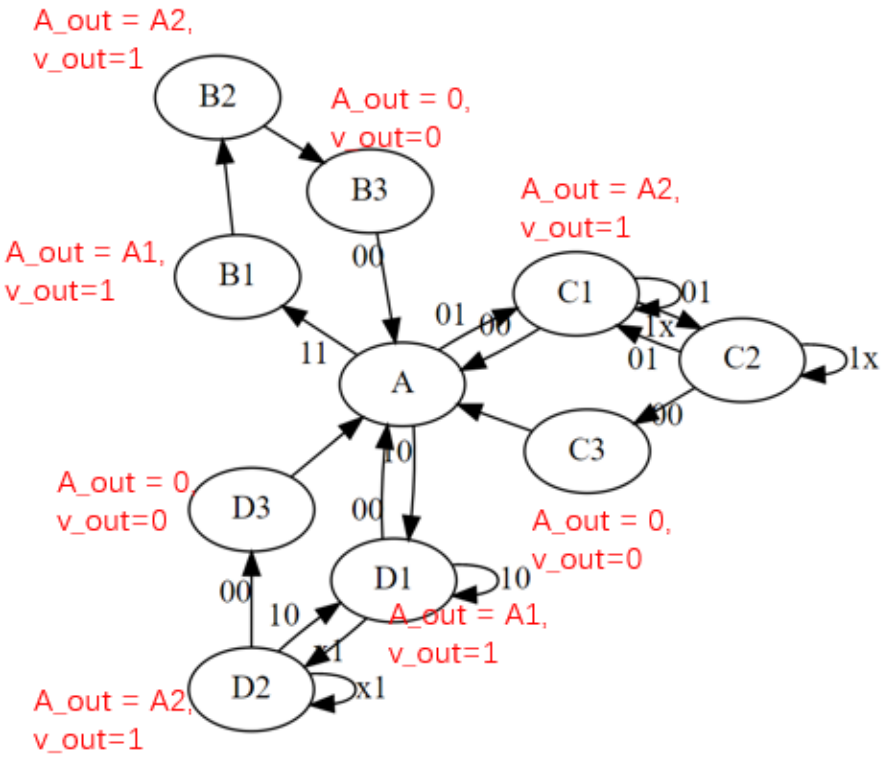


Figure 4.9: The state transition diagram of FSM.

$$t_{v1} + t_{v0} = t_{AER} \tag{4.1}$$

Figure 4.10 (a) shows the extreme case for loop1. The valid1 changes some time prior to valid2. But this time amount is less than the delay value of Muller-C element, so the FSM cannot finish a single state transition in this time period. Then valid2 becomes 1 later and the FSM will enter the B1 state at loop1. In this case, although two valid signals come at different times, they are still treated as simultaneous stimuli for the FSM. To ensure that the FSM can finish loop1 in one transmission cycle, Equation 4.2 must be met.

$$5 \times t_d \leq t_{AER} \tag{4.2}$$

The other two extreme cases may happen in loop2 or loop3. In case 2, shown in Figure 4.10 (b), the valid2 signal first becomes high and the FSM sends bus2's AER signal to the receiver. Then after some time longer than t_d, the valid1 signal also changes to 1, so the FSM turns to the state to sample the first AER address. After that, valid1 changes to low for a short period and then returns to high again, while valid2 keeps high during this period. Therefore, the FSM must be able to finish one state transition to sample bus2 in the short period when bus1 is low, as Equation 4.3 shows. Otherwise, an AER signal from bus2 will be lost.

$$t_{v0} \geq t_d \tag{4.3}$$

The last extreme case happens when the valid1 signal rises between a falling and rising edge of valid2, but the time intervals between two adjacent edges are both less than t_d, so an AER signal from bus2 is not sampled. To prevent this from happening, Equation 4.4 must be ensured. A similar requirement also holds for t_{v1} as Inequality 4.5 shows.

$$t_{v0} \geq 2 \times t_d \tag{4.4}$$

$$t_{v1} \geq 2 \times t_d \tag{4.5}$$

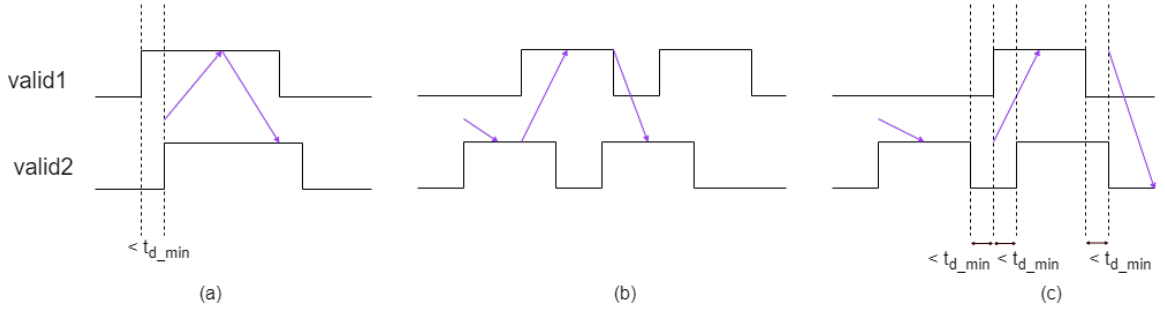


Figure 4.10: Extreme cases for the timing constraints of the FSM

Based on the three extreme cases mentioned above, the delay value of the Muller-C element should meet Inequality 4.6. Besides, the receiver determines the lower bound of the delay value in FSM. Assume that the receiver can only handle the data that lasts longer than t_{r_min}, then inequality 4.7 must be met. The max and min at the end of each item in the inequality mean the maximum and minimum value, considering on-chip variation.

$$t_{d\_max} \leq min\left\{\frac{1}{5}t_{AER\_min}, \frac{1}{2}t_{v0\_min}, \frac{1}{2}t_{v1\_min}\right\} \tag{4.6}$$

$$t_{d\_min} \geq t_{r\_min} \tag{4.7}$$

Another kind of timing issue comes from the state assignment. When a transition happens between 2 states with more than 1 different bit, the FSM may jump into an intermediate state before stabilizing at the final state. This may cause the FSM to enter the wrong state and generate false outputs. To avoid this issue, we need to guarantee that every 2 adjacent states of a transition should only have 1 bit different. So in our solution, each loop has four states, which enables the use of gray code for 2-bit state assignment. Since there are three loops in the FSM, we use 2 bits for each loop encoding, leading to 6 bits in total for the state encoding. In each loop's transition, only 2 bits out of 6 may change, with 1-bit change for a single transition. Through

this kind of race-free state assignment, the glitches caused by state transition can be eliminated.

The FSM is also scalable by adding another one or more states between every 2 adjacent states. The number of states can be increased when the delay value of available Muller-C elements is much smaller than the lower bound of its requirements.

4.3.3 Self-timed shift register

From the discussion of select logic in the previous section, we know that a stop signal can be issued to the transmitter to stop its feedback loop's operation when there are input AER signals both from the corresponding NA and adjacent Fence. However, since the AER signals from the Fence and NA can come at any time, a problem we call "Bus Compression" may appear at the output of select logic. It can possibly influence the function and performance of the Fence.

Taking Figure 4.5 as an example, assume that at time 0, a new set of signals are generated at the output of the transmitter's sampler. The arrival time of valid AER inputs at the Fence is t_{arr_b1} and t_{arr_b2}, respectively (corresponding to the input from the adjacent Fence and NA). The arrival time of the stop signal at the transmitter's MUX in the feedback loop is t_{stop}. And the arrival time of the new feedback value at the sampler's reset ports is t_{arr_feed}. The propagation delay of select logic and control logic are t_{sel} and t_{con}. So t_{arr_b2} can be calculated as Equation 4.8.

$$t_{arr\_b2} = t_{hop} + t_{p2ck} + t_{ck2q} \tag{4.8}$$

Then the arrival time of the stop signal at the transmitter's feedback loop is expressed in Equation 4.9. It conforms to the select logic's function that a stop signal will be generated when both inputs have a valid AER signal.

$$t_{arr\_stop} = max\left\{t_{arr\_b1}, t_{arr\_b2}\right\} + t_{con} + t_{sel} \tag{4.9}$$

Besides, t_{arr_feed} can be calculated in Equation 4.10. Here the propagation of the MUX that performs the stop function is neglected.

$$t_{arr\_feed} = t_{pri} + t_{inv} + t_{delay} + t_{rst2q} \tag{4.10}$$

There are 2 kinds of relationships between t_{arr_b1} and t_{arr_b2}. If $t_{arr\_b1} \leq t_{arr\_b2}$, which means that a valid AER signal from the adjacent Fence comes earlier than that from the NA, then the corresponding NA's transmitter must stop at the current transmission cycle to avoid data loss. Therefore, Inequality 4.11 must be met.

$$t_{hop} + t_{p2ck} + t_{con} + t_{sel} \leq t_{pri} + t_{inv} + t_{delay} \tag{4.11}$$

On the other hand, if $t_{arr\_b1} \geq t_{arr\_b2}$, the AER signal from the NA may first appear at the output of the select logic for a short period before the AER signal from another Fence comes in. In this case, it has a minimum duration time at the select logic's output. Taking Equation 4.8, 4.9, 4.10 and 3.6 into account, the shortest duration time of an AER signal at select logic's output is expressed in Equation 4.12.

$$t_{min\_dur\_1} = t_{inv} + t_{delay} - (t_{con} + t_{sel} + t_{enc}) \tag{4.12}$$

The same conditions also apply to the whole priority encoder solution. Inequality 4.13 must hold for $t_{arr\_b1} \leq t_{arr\_b2}$ condition. For the $t_{arr\_b1} \geq t_{arr\_b2}$ scenario, the minimum duration time is expressed in Equation 4.14.

$$t_{hop} + t_{p2ck} + t_{con} + t_{sel} \leq t_{pri\_enc} + t_{demux} + t_{delay} \tag{4.13}$$

$$t_{min\_dur\_2} = t_{demux} + t_{delay} - (t_{con} + t_{sel} + t_{han}) \tag{4.14}$$

This minimum duration time is much shorter than a normal transmission cycle. So the AER signal is compressed in time. However, the FSM has a lower bound requirement for the duration time of AER signal. Besides, we also need a relatively fixed AER signal width to transmit among different Fences. Therefore, the self-timed shift register is designed to recover AER signal's duration time.

The idea of the self-timed shift register comes from [31] and [32]. They proposed and designed a self-timed FIFO circuit and some of its variations. The self-timed FIFO they proposed is composed of many stages, with adjacent stages communicating by a pair of handshake signals. In our design, no handshake signals are wanted. Besides, our circuit element here doesn't need data processing functions. The only function it should have is to restore the duration time of each AER signal.

Based on the above-mentioned considerations, we design two self-timed shift registers that can restore the AER signal width. The first is shown in Figure 4.11. Its first three stages, which consist of latches, are used to shift the AER signal in sequence. The last stage, a simple delay element, outputs the AER signal with the desired width. There are some feedback loops among different stages. Simply put, the feedback loop determines if this stage is occupied. The term "occupied" means that input and output data are different, which indicates that new data needs to be shifted at this stage. Each latch is transparent if its current stage is occupied and the next stage is not occupied. To ensure the correct function of this circuit, the propagation delay of the latch must be higher than the feedback loop to avoid data loss, as expressed in Equation 4.15. The t_{dq} means the latch's propagation delay from input D to output Q when E is 1. The t_{path1} and t_{path2} represent the delay of two paths marked by purple dash lines, respectively. This design can ensure a minimum duration time of the delay element's value. But there is no upper bound for it.

$$t_{dq} \geq max\left\{t_{path1}, t_{path2}\right\} = t_{NEQ} + t_{AND} + t_{INV} \tag{4.15}$$

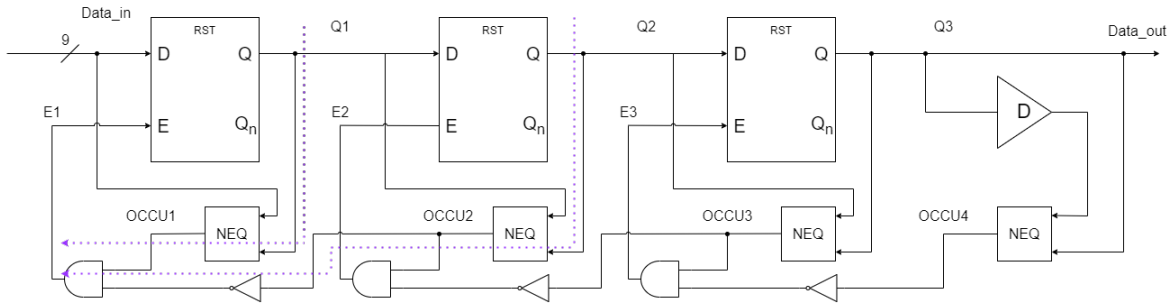


Figure 4.11: The diagram of first self-timed shift register

The timing analysis of this circuit can be broken into stages. Assume that each stage is stable (no changes will happen in the circuit) when a new AER signal comes in. In the first stage, the feedback loop from Q1 to E1 first updates its value. Then this new AER signal could pass through the latch in stage 1. This process repeats at stage 2 and stage 3. So the delay of this self-timed shift register is expressed in Equation 4.16. The latch stages can be increased or decreased according to the input. If the self-timed shift register continuously takes in new AER signals that last much shorter than the desired duration time, there should be more stages to store each AER signal to avoid data loss. In our design, the compressed bus only happens when the stop signal changes from 0 to 1. As a result, we can have fewer stages in our shift register. 2 or 3 stages are enough for our design.

$$t_{st_reg} = 3 \times t_{dq} + 3 \times t_{path1} \tag{4.16}$$

Another self-timed shift register we have designed is shown in 4.12. In this design, the last stage is a latch and a MUX controlled by an asynchronous FSM. The state transition diagram of FSM is shown in 4.13. EX1 and EX2 are inputs that indicate if the final latch is occupied and if the input to the final latch is 0. E and SEL control the final latch and MUX, respectively. In states b and c, the latch is transparent, and MUX outputs the data from the latch. In states d and e, the latch cannot pass data but the MUX still outputs data from the latch. In these 2 states, the final stage becomes unoccupied and can accept data from the previous stage again. This design of self-timed latch ensures a fixed length of output data width except 0. However, it's much more challenging to design and adds more area overhead due to its complex last stage. In our scenarios, the problem is that the duration time of the AER signal is compressed. And there are no requirements for the upper bound on data duration time. Therefore, the first design can meet the needs for data transmission.

4.3.4 Replace FSM with a 2-bit tiny transmitter

There exists another method to send forward and backward AER signals to the receiver in sequence, which is to use a transmitter structure that has been proposed in Chapter 3.

Figure 4.14 shows the structure of a 2-bit tiny transmitter. It contains a sampler, a 2-bit prioritizer, and an output multiplexer. The sampler's input is 2 valid signals

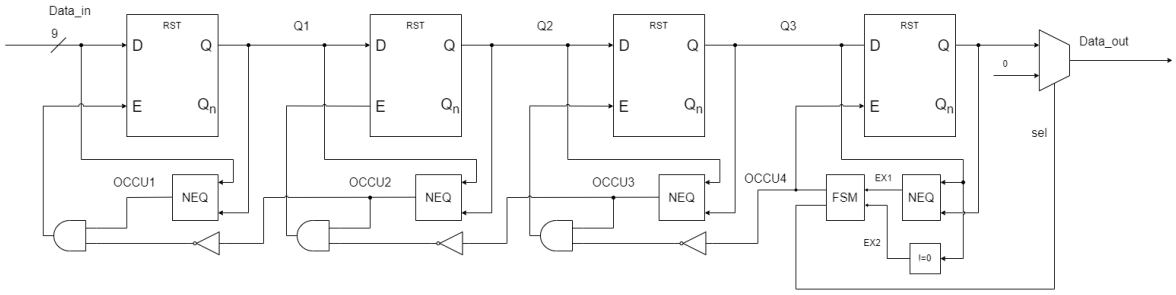


Figure 4.12: The diagram of second self-timed shift register

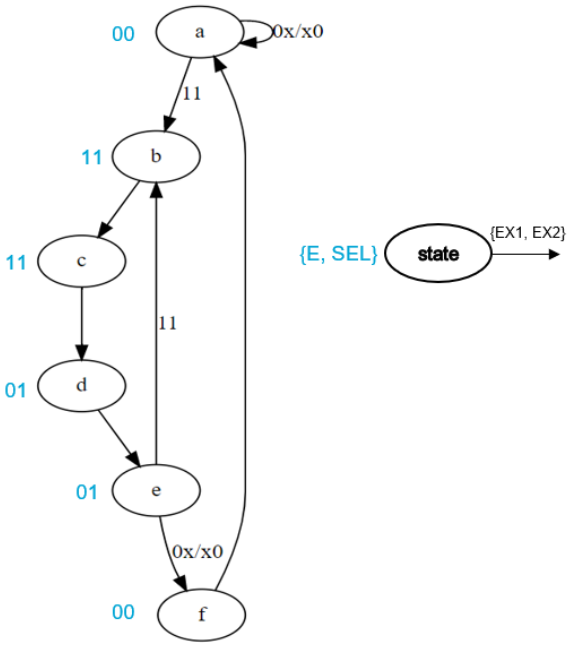


Figure 4.13: The state transition diagram of FSM in the self-timed shift register

from the forward and backward bus. The output of the 2-bit prioritizer determines the output of MUX.

Inequality 4.17 indicates the timing constraint for the delay value of the delay element in the feedback loop. t_{trans} represents the transmission cycle for an AER signal, which is determined by NAs' transmitter. To send the concurrent AER signals from the forward and backward bus, a transmission to the receiver must be finished in half of the transmission cycle. Inequality 4.18 shows another constraint for the delay element's inertial delay. It is used to resolve the glitches caused by close input valid signals.

$$t_{pri} + t_{inv} + t_{delay} + t_{rst2q} \leq \frac{1}{2} \times t_{trans} \tag{4.17}$$

$$t_{delay\_iner} \geq max\left\{t_{pri}, t_{mux}\right\} \tag{4.18}$$

The structure of the Fence that uses a 2-bit transmitter to send AER signals to the

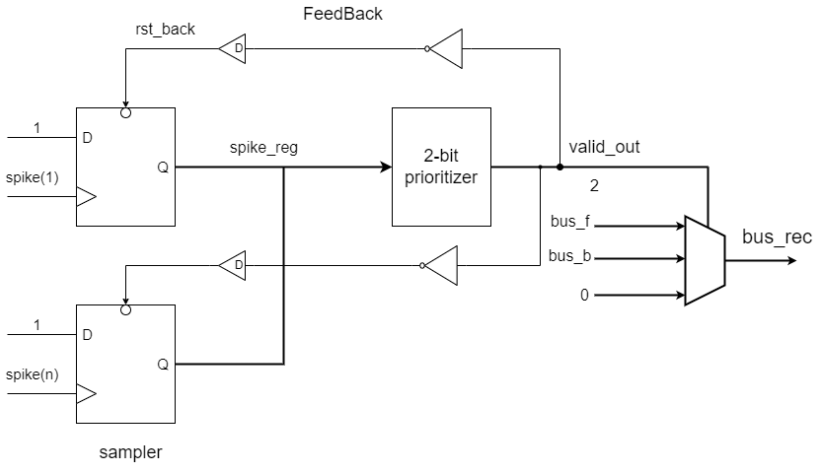


Figure 4.14: The 2-bit tiny transmitter structure

receiver is shown in Figure 4.15. The 2-bit transmitter scheme has a less complicated control algorithm. It also has a loose timing constraint compared with asynchronous FSM. It can use up to half transmission cycles to send data, while in the FSM scheme, the minimum duration time of output AER signal should be less than $\frac{1}{5}$ of the transmission cycle.

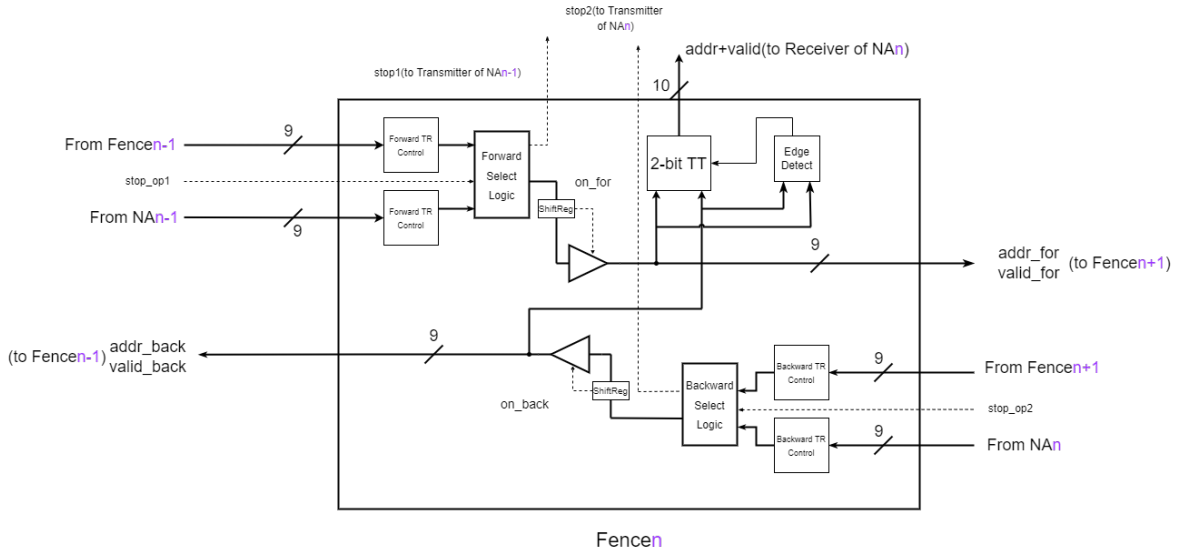


Figure 4.15: The structure of Fence using the 2-bit transmitter at the output to receiver

4.4 Summary

This chapter proposes and implements a segmented bus structure for multi-array communication. A new circuit element called Fence is created to control the data flow of the system. Each component of the Fence is carefully designed to meet the timing requirement.

5 Results and Discussions

In this chapter, the results of the previously designed interconnect are presented. Firstly, the functional simulation results are shown for both point-to-point interconnect and segmented-bus structures. Then, the synthesis results of part of the design are listed.

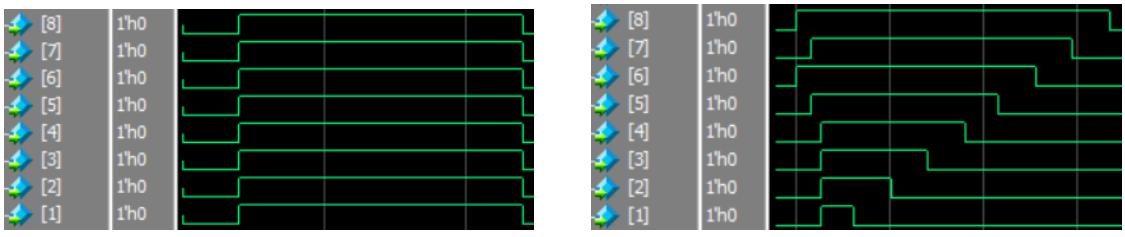
5.1 Functional Simulation

To test the function of our interconnect structure, some spike patterns are generated in our testbench. Then the waveform at each stage is checked to ensure that the interconnect operates correctly and all the timing issues are solved by our proposed methods. The proposed design is implemented in 64-bit, 256-bit, and 1024-bit. To ease the illustration, we use a 64-bit interconnect with limited input to show the waveform.

5.1.1 Transmitter simulation

The transmitter is the focused part of point-to-point communication. At the functional simulation stage, we assume some delay values for each circuit component. The delay value of the delay element is chosen based on Inequalities in Chapter 3.

Figure 5.1 (a) shows the waveform, the input spikes, and the sampler's output. The input is some close and simultaneous spikes. Then each flip-flop samples the input spikes, respectively, as Figure 5.1 (b) shows. Each bit is reset after it has been encoded and sent out by the transmitter. Note that the spike width is much higher than circuit delays. Therefore we zoom out the input spikes so that it fits the window.



(a) (b)

Figure 5.1: The waveform of (a) input spikes (b) output of sampler

Figure 5.2 shows the waveform from the tree encoder and the following address handling circuit. The Addr_in signal represents the signal from the tree encoder, and the Addr-enc is the AER signal after adding 1 and appending the Neuron Array index. Since 0 is rendered useless in AER communication, a plus 1 operation is always needed

when the enable signal is 1. There are some glitches at the start of the AER signal. They are eliminated by the output flip-flop and feedback loop.

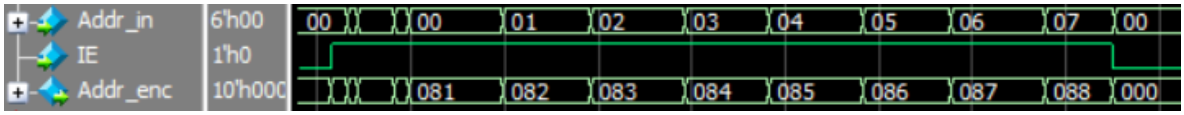
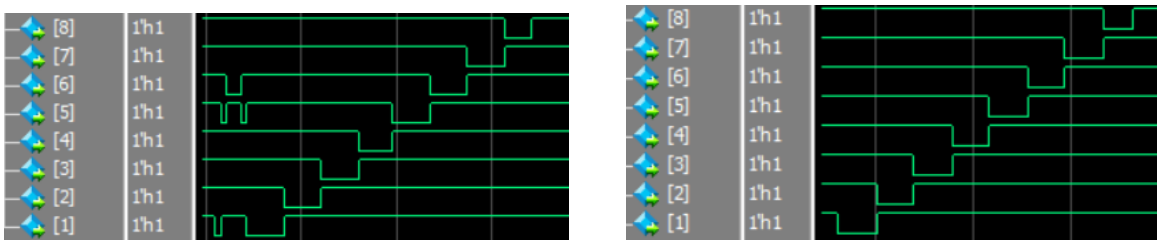


Figure 5.2: Waveform from tree-encoder and following address-handling unit

Figure 5.3 (a) shows the waveform of the demux's output. The demux decodes the $log_2 n$ bit address into n bit lines that feedback to the sampler. There are some glitches at some of the bit lines of the demux's output. However, since they last short than the delay element's inertial delay, they are filtered out at the output of the delay element.



(a) (b)

Figure 5.3: The waveform of (a) demux outputs (b) output delay element

Figure 5.4 shows the waveform of the transmitter's output. The transmitter's output is driven by an event signal called clk in the Figure. By using this event-driven flip-flop, the previous glitches at the address line are filtered out. After all the AER signals have been set, the output is reset by the address line. The time between two yellow cursors is a transmission cycle.

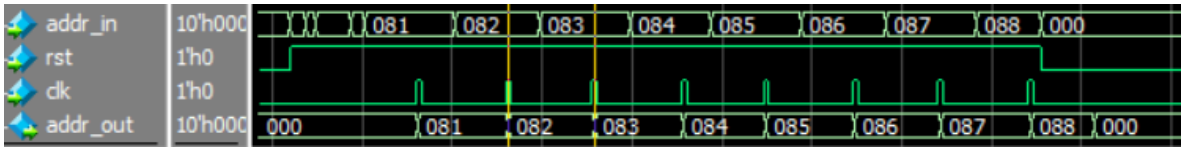


Figure 5.4: Waveform of the transmitter's output

In the waveforms shown above, the priority of AER signals decreases from high to low. This is because these waveforms are taken from the beginning. The flip-flops in the priority encoder are reset at the start of the simulation. A rotating priority can be seen after the interconnect has worked for some time with irregular spike patterns, as Figure 5.5 shows.

The waveform of the transmitter, which is composed of a separated prioritizer and one-hot encoder, is somewhat similar to the above one. The difference is that it just uses the prioritizer's output to give feedback to the sampler's reset port directly.

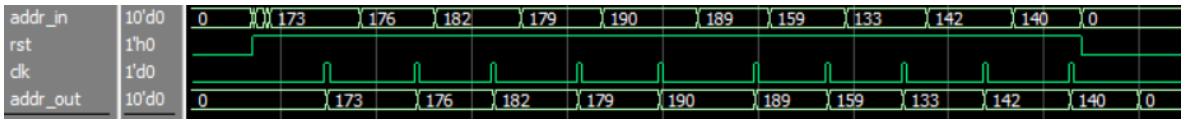


Figure 5.5: The rotating priority effect

5.1.2 Segmented-bus simulation

After simulating the point-to-point interconnect, we also simulate the segmented-bus structure for multi-array communication. The focus of multi-array simulation falls on the behavior and waveform of the Fence.

Figure 5.6 shows the waveform of a Fence's forward path. The first 2 rows represent the input AER signal from the adjacent Fence and corresponding NA. Then after a certain delay, some AER signal passes to the select logic, while others are refused by the control logic. The input AER signals of select logic are bus_for_1 and bus_for_2. We can see that when there are non-zero AER signals at both of the select logic's inputs, a stop signal is issued, and the NA's output keeps at the same value until AER signals from the Fence have temporarily completed transmission. And the output of select logic, indicated by bus_for_in in the figure, is a sequence of all the AER signals that pass the control logic. We can see that signal 0fe has a less duration time than others. After passing the self-timed shift register, all the AER signals at the input of the tri-state gate have a fixed duration time. Then after the tri-state gate, it passes to the next Fence and goes to the FSM to be sent to the corresponding NA's receiver.

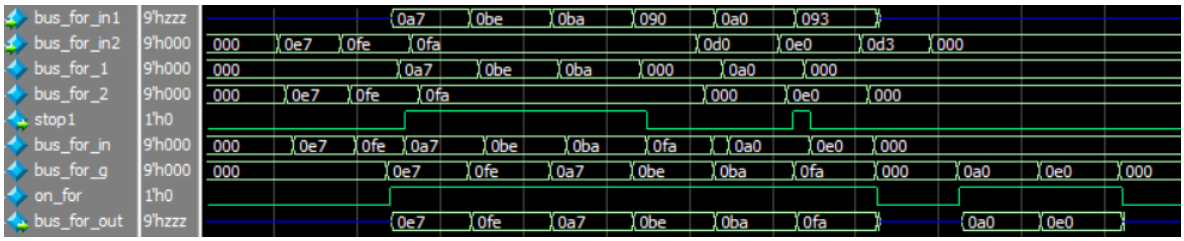


Figure 5.6: Waveform of Fence

Figure 5.7 shows the waveform at the output of the Fence. This output is connected to an NA's receiver. From the Figure, we can see that the asynchronous FSM successfully sends the AER signals from forward and backward output in sequence. Although their duration time is not fixed, considering that these AER signals will be decoded but not propagated to other stages. This is not an issue.

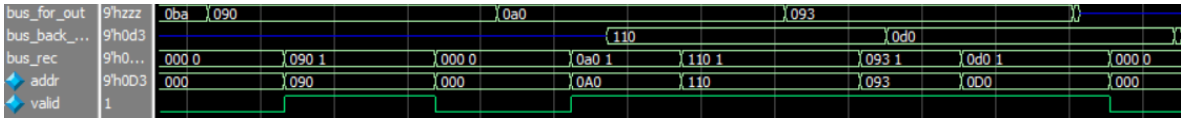


Figure 5.7: Waveform of the output of the Fence to the receiver

Figure 5.8 also shows the waveform of the Fence's output but using a 2-bit transmitter. Through this method, the duration time of each AER signal at the output is relatively fixed.

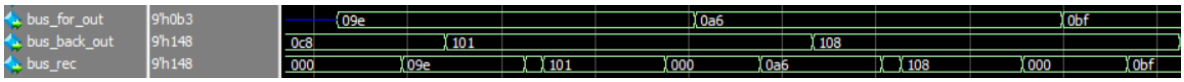


Figure 5.8: Waveform of Fence's output by a 2-bit transmitter

5.2 Synthesis Results

Synthesis is also partially done(except for the delay element) for the transmitter and Fence.

5.2.1 Transmitter

The core component of the transmitter is the priority encoder. Therefore different priority encoders with different input bits are synthesized first.

Table 5.1 shows the synthesis results of the priority encoder. From the table, we can get the following 2 conclusions:

- Firstly, as for the 64-bit prioritizer, our proposed structure has a similar propagation delay to the normal prioritizer, which does not have a priority rotation function. However, it comes at a price of much more area overhead. The same applies to the 64-bit priority encoder.
- Secondly, for the same bit structure, the prioritizer scheme and the whole priority encoder scheme have a similar propagation delay. However, the whole priority-encoder scheme saves more than 20% area and more than 40% nets, count. Considering the fact that the priority encoder or prioritizer needs to collect and process signals from different neurons which are much larger in size, the wire delay may dominate the delay composition, and wire congestion could be a potential problem in place and route. Therefore, the nets count here is also an important criterion for choosing the transmitter structure. Based on the above information, the whole priority encoder is a better choice for building the transmitter.

Table 5.1: Synthesis results of different schemes for different bits

Structure	Propagation Delay/ns	Area	Nets Inside	Priority rotation
64-bit normal tree prioritizer	0.3328	517.2439	1768	No
proposed 64-bit tree prioritizer	0.3400	920.7100	2588	Yes
64-bit normal tree priority encoder	0.3255	326.9280	910	No
proposed 64-bit tree priority encoder	0.3313	678.5520	1808	Yes
proposed 256-bit tree prioritizer	0.3829	3763.1020	12033	Yes
proposed 256-bit tree priority encoder	0.3766	2961.3640	7539	Yes
proposed 1024-bit tree prioritizer	0.4228	12783.6099	55008	Yes
proposed 1024-bit tree priority encoder	0.4126	9375.1699	31006	Yes

Having chosen the proper priority encoder, we synthesize the transmitter with a whole priority encoder structure. The synthesis is done with 64, 256, and 1024 input bits, respectively. The synthesis excludes the delay elements in the feedback loop.

Table 5.2 shows the synthesis results of transmitters with different input bits. The fourth column is the delay value we chose for the feedback delay element. Based on the constraints proposed in Chapters 3 and 4, the delay value is chosen in a conservative way, which is at least 1.5 times higher than the priority encoder's delay and higher than the total path delay except for the delay element itself. The fourth column shows the Inertial delay required for the delay element. Any pulses that last shorter than this value will be filtered out. The throughput is measured based on how many spikes it can transmit per unit of time.

Table 5.2: Synthesis results for the transmitter(delay element excluded)

Structure	Propagation Delay/ns	Area	Delay value choice/ns	Inertial Delay/ns	Throughput/Gbps
With proposed 64-bit tree priority encoder	0.3924	1205.3019	0.5	0.35	1.12
With proposed 256-bit tree priority encoder	0.5254	4964.4839	0.6	0.4	0.89
With proposed 1024-bit tree priority encoder	0.5950	16753.7859	0.7	0.45	0.77

We do a post-synthesis simulation to see the delay information and if our transmitter functions correctly. Figure 5.9 shows part of the waveforms of the simulation. The first and second signal is the priority encoder and address handling unit's output. The third one is the event-driven clock signal and the last signal is the final AER output. In post-synthesis simulation, the glitches become much more severe than in functional simulation. Here we guarantee a stable output by event-driven flip-flops introduced in Chapter 3.

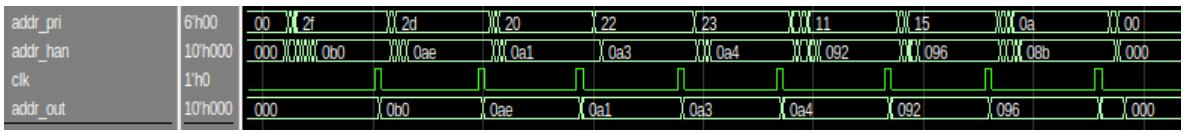


Figure 5.9: Partial waveform of post-synthesis simulation

The transmission cycle not only affects the timing error introduced by the transmitter for each spike, but also influences the number of stages we should use in the self-timed shift register. The priority encoder we designed has a tree structure. Since not all the stages change their state each time when they encode a spike, there can be some variations in the priority encoder's delay, thus leading to variations in the transmission cycle. We input around 1000 spikes, which include plenty of simultaneous spikes, to measure the delay distribution for the 64-bit, 256-bit, and 1024-bit transmitter.

Figure 5.10 shows the histogram of different transmitters' delay distribution. We use the worst-case delay as a reference, and each bar represents the proportion of spikes that has a transmission cycle in a certain range. For each transmitter, the transmission cycles mostly fall in the range of 80% to 100% of worst-case delay. Based on the synthesis results, the variation of the transmission cycle is within 0.2 ns to 0.3 ns. There are also some transmissions that last shorter than 60% of the worst-case delay. This happens when all the spikes have been transmitted and the AER output returns to

0. The transmission cycle can be restored by the self-timed shift register in the Fence.

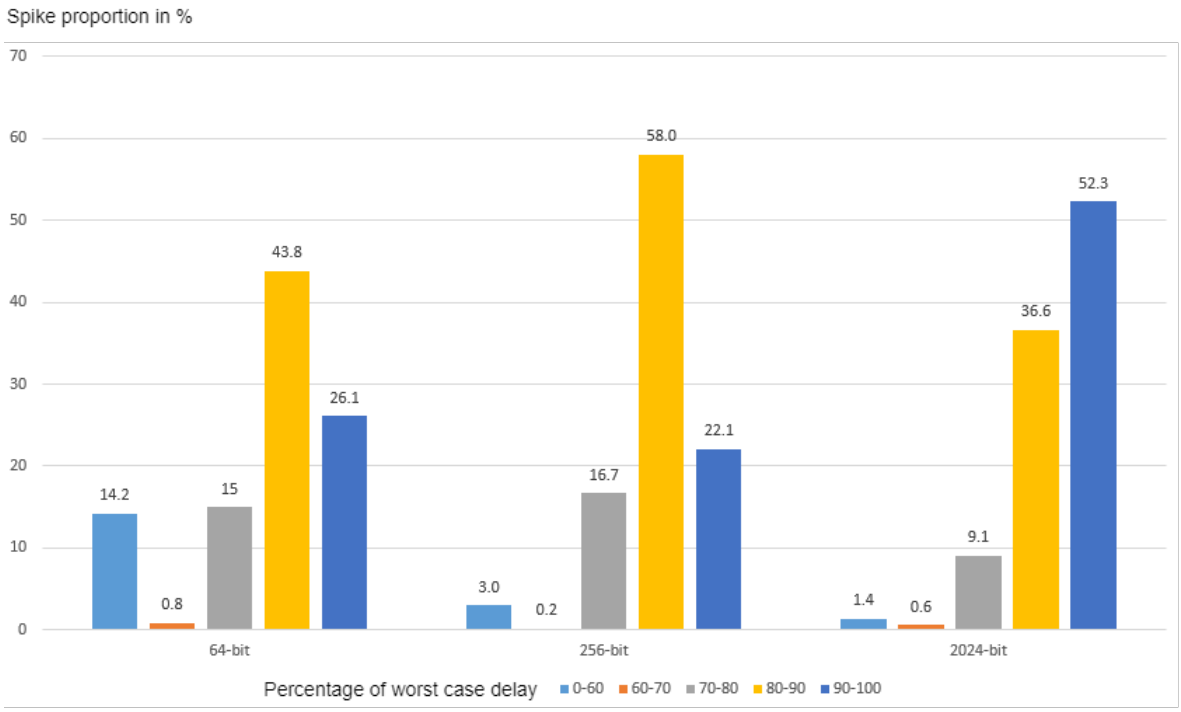


Figure 5.10: The delay distribution of transmitters

5.2.2 Fence

We also partially synthesize the Fence. Unlike the transmitter with delay elements only in the feedback loop, the Fence contains delay elements in most sub-blocks. So we break down each of its sub-blocks and synthesize them.

Table 5.3 shows the synthesis results of the components in Fence. All the sub-blocks have 9 input AER signal wires. The self-timed shift register has a higher delay than other components since it has multiple states. In each stage, it needs to wait for the feedback's delay and the latch's delay before it can pass to the next stage. The FSM, under this condition, may not operate correctly sometimes. This is because its delay of next state forming logic is higher than the required one by Inequality 4.6. As a result, its delay element should have a 0 delay value which is not possible. Therefore, using the 2-bit transmitter to send data to the receiver is a safer option. Besides, it also brings area benefits compared with asynchronous FSM.

The sum of these elements' area is the area for one path. The Fence consists of 2 paths. The Fence delay from the input to the output is the sum of control logic, select logic, self-timed shift register, and tri-state gate's delay. The control logic is not synthesized here because it is related to the structure of specific SNNs. Table 5.4 shows the delay and area of the Fence with 9, 11, and 13-bit input address lines.

Table 5.3: Synthesis results for the Fence's sub-blocks(delay element excluded)

Component	Delay	Area/um^2
select logic	0.2667	74.782
self-timed shift register	0.6544	81.438
FSM	0.2265 for next state forming; 0.2899 for output forming	121.520
tri-state gate	0.2740	74.088
2-bit Transmitter	0.2332	91.336

Table 5.4: Synthesis results for the Fence in total(delay element excluded)

# Address Lines	Delay	Area/um^2
9	1.1951	551.952
11	1.1869	705.502
13	1.2008	839.664

5.3 Discussion

Two criteria are critical in the segmented-bus structure: throughput and delay. The throughput of the interconnect is determined by the transmitter in each NA and the control logic in the Fence. The transmitter determines the peak throughput for each NA. As for the control logic, it determines which AER signal can pass and which cannot. Generally speaking, if the control logic passes all the AER signals so that the Fence needs to send a stop signal to NAs continuously, the throughput will be decreased.

The delay of the circuit is determined by the transmitter and the Fence. Since the transmitter sends AER signals in sequence, it adds extra delay to simultaneous spikes. As for the Fence, all components add delay to the AER signal. The AER signals in the Fence are used mostly as a bunch of wires that can or cannot pass. It does not have processing utilization. As a result, more input bits only add to the area but have no influence on the delay. Besides, for an AER signal, the more Fences it needs to pass, the more delay it will be added.

The synthesis results are also influenced by the synthesis strategy. In this project. we break the design into delay elements and parts without delay elements. The synthesis is done for parts without delay elements. Another reason for the breakdown is that the tool cannot analyze timing for some circuits, such as the self-timed shift register. The breakdown of the design may hurt synthesis results because the tool cannot perform optimization between different parts. Another factor that has a great effect on synthesis results is the output load in constraints. Here we use 1pf for output load. In the timing report, there is always a steep increase in the delay at the output port (sometimes larger than 50%). Therefore, the interconnect may have a much better performance when using smaller loads.

6 Conclusion and Future work

This chapter first restates the background of the thesis. Then it summarizes the main work we have done in this project and gives a conclusion. Finally, it gives some future aspects that can be done to extend the topic.

6.1 Conclusion

Spiking Neural Networks use spikes to transmit information. The spikes themselves contain timing information for the following Neurons to process. Address Event Representation is the most commonly used communication protocol among different Neuron Arrays. To reduce the power consumption caused by a clock signal and save the area overhead, this thesis aims to design a self-timed interconnect for Neuron Array communication. The self-timed circuit uses the feature of the event itself to start or terminate a transmission. It should have few control signals and is built up on delay elements, latches, flip-flops, and combinational logic. Due to the lack of clock and global control signals, solving the timing problem is the critical part of self-timed circuits.

In this thesis, we start from the point-to-point interconnect for SNNs and focus on its transmitter design. Generally speaking, the transmitter samples the edge of each spike and feedback a reset signal to complete a transmission cycle. The priority encoder is the key component of the transmitter because it needs to handle simultaneous spikes and minimize the delay. Two types of priority encoders are designed in this thesis based on the features of Neuron Arrays. While both the prioritizer with one-hot encoder and the whole priority encoder solution have a similar delay and throughput, the whole priority encoder outperforms the other one in terms of area and wire counts. Based on the structure we designed, different timing issues are analyzed and some measures are proposed to solve timing problems. Among them, setting the delay element a certain amount of inertial delay as well as adding an event-driven flip-flop at the output of the transmitter is considered the safest.

Besides the point-to-point interconnect, a general segmented-bus topology is proposed and designed for multi-array communication. The segmented bus can connect or isolate different NAs at different times. To control the segmented bus, we designed a structure called Fence. The Fence is responsible for communicating different AER signals among NAs. Due to the lack of a clock signal, the Fence needs to control the timing carefully. It needs to guarantee a fixed duration time for each incoming data and send them to corresponding NAs regardless of their phase relationship. Based on Fences, the AER signal can communicate among NAs in an ordered way.

6.2 Future Work

This thesis realizes a generalized interconnect structure for point-to-point and multi-array communication. Further work can be done in the following aspects.

Firstly, in the point-to-point interconnect structure, we impose a set of constraints on the delay elements. They are left as black boxes in this design. These delay elements can be implemented at transistor level based on our proposed requirements. Besides, the rotated priority solution comes at a price of more area overhead. Based on different SNNs, whether the priority rotation function should be removed to save area is another topic for research.

For multi-array interconnect, out segmented-bus topology is a generalized structure suitable for different kinds of Neuronal communication systems. When it comes to specific SNNs, the structure can be optimized to achieve better performance. For example, how to map a given SNN in this segmented-bus topology to achieve better throughput and power consumption could be studied in the future. Besides, different NAs may be also put in a single segment to decrease the amount of traffic.

Furthermore, this self-timed interconnect is not compatible with the existing synthesis flow. In this work, our strategy is to separate the combinational part from other parts that contain flip-flops, latches, or delay elements. Then, each part is constrained and synthesized on its own. The final results of different parts are combined again. However, this will weaken Design Compiler's optimization effect, so the results may not be the best. Further research can be done to make self-timed circuits suit the traditional synthesis flow.

Bibliography

[1] S. A. Mohamed, M. Othman, and M. H. Afifi, "A review on data clustering using spiking neural network (snn) models," *Indones. J. Electr. Eng. Comput. Sci*, vol. 15, no. 3, p. 1392, 2019.

[2] A. Saxena, M. Prasad, A. Gupta, *et al.*, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.06.053. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217311815.

[3] M. A. Mahowald, "The address-event representation communication protocol. aer 0.02," *California Institute of Technology,. Pág*, 1993.

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

[5] P. Falez, P. Tirilly, I. M. Bilasco, P. Devienne, and P. Boulet, "Mastering the output frequency in spiking neural networks," in *2018 international joint conference on neural networks (IJCNN)*, IEEE, 2018, pp. 1–8.

[6] M. R. Jan, C. Anantha, N. Borivoje, *et al.*, "Digital integrated circuits: A design perspective," *ed: Prentice Hall Upper Saddle River, NJ*, 2003.

[7] E.-G. Merino Mallorqui, "Digital system for spiking neural network emulation," B.S. thesis, Universitat Politècnica de Catalunya, 2017.

[8] R. Chen, N. Vijaykrishnan, and M. Irwin, "Clock power issues in system-on-a-chip designs," in *Proceedings. IEEE Computer Society Workshop on VLSI '99. System Design: Towards System-on-a-Chip Paradigm*, 1999, pp. 48–53. DOI: 10.1109/IWV.1999.760472.

[9] M. Shams, J. C. Ebergen, and M. I. Elmasry, "Asynchronous circuits," *John Wiley's Encyclopedia of Electrical Engineering*, pp. 716–725, 1999.

[10] E. Nigussie, S. Tuuna, J. Plosila, P. Liljeberg, J. Isoaho, and H. Tenhunen, "Boosting performance of self-timed delay-insensitive bit parallel on-chip interconnects," *IET circuits, devices & systems*, vol. 5, no. 6, pp. 505–517, 2011.

[11] M. Miller, C. Segal, D. Mc Carthy, A. Dalakoti, P. Mukim, and F. Brewer, "Impolite high speed interfaces with asynchronous pulse logic," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 99–104.

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

[13] P. A. Beerel, R. O. Ozdag, and M. Ferretti, *A designer's guide to asynchronous VLSI*. Cambridge University Press, 2010.

[14] S. Divekar and A. Tiwari, "Multichannel amba ahb with multiple arbitration technique," in *2014 International Conference on Communication and Signal Processing*, IEEE, 2014, pp. 1854–1858.

[15] S. Pande, F. Morgan, G. Smit, *et al.*, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," *Parallel computing*, vol. 39, no. 9, pp. 357–371, 2013.

[16] T. Seceleanu, V. Leppanen, J. Suomi, and O. S. Nevalainen, "On the organization of multisegmented bus," *Turku Centre for Computer Science, TUCS Technical Reports*, no. 647, 2004.

[17] T. Seceleanu, J. Plosila, and P. Lijeberg, "On-chip segmented bus: A self-timed approach [soc]," in *15th Annual IEEE International ASIC/SOC Conference*, IEEE, 2002, pp. 216–220.

[18] A. Balaji, Y. Wu, A. Das, F. Catthoor, and S. Schaafsma, "Exploration of segmented bus as scalable global interconnect for neuromorphic computing," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 495–499.

[19] A. Hore, S. Panda, A. Chakraborty, S. Bandyopadhyay, and S. Chakrabarti, "Effects of spike width on spiking frequency in a cmos neuron design following a subthreshold approach," in *2021 Advanced Communication Technologies and Signal Processing (ACTS)*, IEEE, 2021, pp. 1–6.

[20] C. Kun, S. Quan, and A. Mason, "A power-optimized 64-bit priority encoder utilizing parallel priority look-ahead," in *2004 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, vol. 2, 2004, pp. II–753.

[21] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "An fpga approach for high-performance multi-match priority encoder," *IEICE Electronics Express*, pp. 13–20 160 447, 2016.

[22] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A scalable high-performance priority encoder using 1d-array to 2d-array conversion," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 9, pp. 1102–1106, 2017.

[23] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, "A 219-μw 1d-to-2d-based priority encoder on 65-nm sotb cmos," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2018, pp. 1–4.

[24] K. A. Chandra and N. Murty, "Low power high performance priority encoder using 2d-array to 3d-array conversion," *Procedia Computer Science*, vol. 171, pp. 1037–1045, 2020.

[25] A. Ojha and R. Mehra, "Low power layout design of priority encoder using 65nm technology," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 23, no. 9, 2015.

[26] K. M. Ali, H. Mostafa, and T. Ismail, "High performance layout-friendly 64-bit priority encoder utilizing parallel priority look-ahead," in *5th International Conference on Energy Aware Computing Systems & Applications*, IEEE, 2015, pp. 1–4.

[27] J. Schemmel, A. Grubl, K. Meier, and E. Mueller, "Implementing synaptic plasticity in a vlsi spiking neural network model," in *The 2006 ieee international joint conference on neural network proceedings*, IEEE, 2006, pp. 1–6.

[28] J. C. Barros and B. W. Johnson, "Equivalence of the arbiter, the synchronizer, the latch, and the inertial delay," *IEEE Transactions on Computers*, vol. 32, no. 07, pp. 603–614, 1983.

[29] J. Juan-Chico, P. R. de Clavijo, M. J. Bellido, A. J. Acosta, and M. Valenia, "Inertial and degradation delay model for cmos logic gates," in *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, vol. 1, 2000, pp. 459–462.

[30] R. F. Tinder, "Asynchronous sequential machine design and analysis: A comprehensive development of the design and analysis of clock-independent state machines and systems," *Synthesis Lectures on Digital Circuits and Systems*, vol. 4, no. 1, pp. 1–236, 2009.

[31] H. Bahbouh, A. E. Salama, and A. Khalil, "The design and simulation of fifo using self-timed based asynchronous circuit elements," in *Proceedings of the Fifteenth National Radio Science Conference. NRSC'98 (Cat. No. 98EX109)*, IEEE, 1998, pp. C5–1.

[32] E. Brunvand, "Low latency self-timed flow-through fifos," in *Proceedings Sixteenth Conference on Advanced Research in VLSI*, IEEE, 1995, pp. 76–90.