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

CAS-2021-4993632

M.Sc. Thesis

Physical Characterization of Asynchronous Logic Library

Pai LI

Abstract

Neuromorphic electronic systems have used asynchronous logic combined with continuous-time analog circuits to emulate neurons, synapses, and learning algorithms. It is attractive because of its low power consumption and feasible implementation. Typically, the neuron firing rates are lower than the modern digital systems. Thus, the endpoints of neuromorphic electronic systems are clusters of neurons instead of individual neurons. Address event representation (AER) was proposed in 1991 to multiplex communication for a cluster of neurons into an individual communication channel. AER circuits provide multiplexing/demultiplexing functionality for spikes that are asynchronously generated by/delivered to an array of individual neurons. Asynchronous techniques are not only used in neuromorphic electronic systems, but also widely used in globally asynchronous and locally synchronous (GALS) SoCs, or SoCs with full-asynchronous solutions.

However, commercial tools on the market do not support designing asynchronous circuits, making the circuits cannot be adopted easily by most products. This thesis aims at addressing the challenge by providing an asynchronous library establishment strategy. The strategy uses SR-latches as standard asynchronous cells together with logic gates to build an AER communication circuit. With the strategy, the performance of using a modified traditional arbiter in the AER transmitter can be compared favourably with using state-of-the-art arbiters. A back-end flow and a verification flow are developed to evaluate the performance of the design as well as to check the feasibility of the strategy. The proposed 32-bit AER transmitter under TSMC 28nm CMOS technology sacrifices area and power to achieve better timing performance, where the modified arbiter inside has an **11.54%** better response time than the arbiter who used to be the best in an old comparison.

Physical Characterization of Asynchronous Logic Library

A Design of AER Transmitter and Its Characterization and Back-end Design Flow

---

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Electrical Engineering

by

Pai LI
born in Jilin, China

This work was performed in:

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

Delft University of Technology

Delft University of Technology
Department of
Microelectronics & Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Physical Characterization of Asynchronous Logic Library"** by **Pai LI** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 23rd August, 2021

Chairman: _____
Dr.Ir. Rene van leuken

Advisor: _____
Dr. Amir Zjajo

Committee Members: _____
Dr.Ir. Sorin Cotofana

_____

Abstract

Neuromorphic electronic systems have used asynchronous logic combined with continuous-time analog circuits to emulate neurons, synapses, and learning algorithms. It is attractive because of its low power consumption and feasible implementation. Typically, the neuron firing rates are lower than the modern digital systems. Thus, the endpoints of neuromorphic electronic systems are clusters of neurons instead of individual neurons. Address event representation (AER) was proposed in 1991 to multiplex communication for a cluster of neurons into an individual communication channel. AER circuits provide multiplexing/demultiplexing functionality for spikes that are asynchronously generated by/delivered to an array of individual neurons. Asynchronous techniques are not only used in neuromorphic electronic systems, but also widely used in globally asynchronous and locally synchronous (GALS) SoCs, or SoCs with full-asynchronous solutions.

However, commercial tools on the market do not support designing asynchronous circuits, making the circuits cannot be adopted easily by most products. This thesis aims at addressing the challenge by providing an asynchronous library establishment strategy. The strategy uses SR-latches as standard asynchronous cells together with logic gates to build an AER communication circuit. With the strategy, the performance of using a modified traditional arbiter in the AER transmitter can be compared favourably with using state-of-the-art arbiters. A back-end flow and a verification flow are developed to evaluate the performance of the design as well as to check the feasibility of the strategy. The proposed 32-bit AER transmitter under TSMC 28nm CMOS technology sacrifices area and power to achieve better timing performance, where the modified arbiter inside has an **11.54%** better response time than the arbiter who used to be the best in an old comparison.

Acknowledgments

I received a great deal of support and assistance during the writing of this dissertation.

I would first like to express my deep gratitude to my college supervisor, Dr.Ir. Rene van leuken, and the leader of my company supervisor team, Dr. Amir Zjajo, whose invaluable insight into the research questions and methodology, and constructive advice, pushed me to sharpen my thinking and brought my work to a higher level.

I would like to thank my supervisor team from my internship at Innatera Nanosystems for their patient guidance, practical suggestions, and enthusiastic encouragement. I would particularly like to single out two of my company supervisors, Jinbo Zhou and Kamlesh Kumar Singh. I want to thank you for your unrelenting support and for all of the opinions I was given to further my research.

I would also like to acknowledge my colleagues and support staffs from the Circuits and Systems group. You provided me with software support and helped me with useful information about completing my dissertation.

Finally, I would like to thank my parents and friends for their companionship and psychological counselling, especially during this pandemic and lockdown time. Without your support and happy distractions, I could not have completed this dissertation. You are always there for me.

Pai LI
Delft, The Netherlands
23rd August, 2021

Contents

List of Figures

List of Tables

1 Introduction

When no clock is utilized to implement sequencing in a digital circuit, the circuit is asynchronous. These circuits are also known as clockless circuits [13]. Engineers have a long history of research on asynchronous circuits. As early as ILLIAC in 1952 and ILLIAC II in 1962, University of Illinois had claimed that the two computers combine synchronous and asynchronous technologies [14]. A very famous asynchronous microprocessor is MiniMIPS, an asynchronous version of the 32-bit MIPS R3000 microprocessor designed by the Caltech group [15]. Its performance is as fast as four times that of its clocked version in the same technology [16]. Albeit the fact that asynchronous design has the potential of faster speed, lower power consumption, and better modularity in a large system, developing asynchronous circuits is a tough task at the time since there is no available asynchronous circuits designing tool. Alain Martin's communicating hardware processes (CHP) design methodology aided in the development of general asynchronous circuits [13, 17]. Starting with a high-level definition of the intended system, this technique entails the application of a sequence of program decompositions and transformations. The designed circuit will be a valid logical implementation of the specifications, as each step retains the logic of the original program.

Address event representation (AER) communication protocol transfers spikes between bio-inspired chips. It was first proposed together with the early neuromorphic circuits by Sivilotti [18] and Mahowald [19]. These designs were affected by the research in asynchronous circuits design at the time within their laboratory groups at Caltech. With the facilitation of the CHP design methodology, Boahen [20] proposed the first AER communication transistor blocks.

Figure 1.1 illustrates the principle of AER communication. There are cell arrays in the transmitter and receiver. Each cell contains an oscillator that generates pulses of minimum width. These spikes represent events. When generating events, the cell communicates with the periphery and its address is encoded on the external digital bus. Handshaking channels (request and acknowledgement) are used for completing the communication. In the receiver, the cell whose address is on the bus will be pulsed so that the same pulse stream will be delivered to cells with the same address in the receiver from the transmitter.

AER communication circuits are asynchronous because spikes are asynchronous. Thus, to build AER blocks for neuromorphic electronic systems, the problem of designing asynchronous circuits shall be solved.

1.1 Problem Statement

Many examples show that using asynchronous circuits to design custom chips can achieve cutting-edge performance and/or power consumption. But generally, there

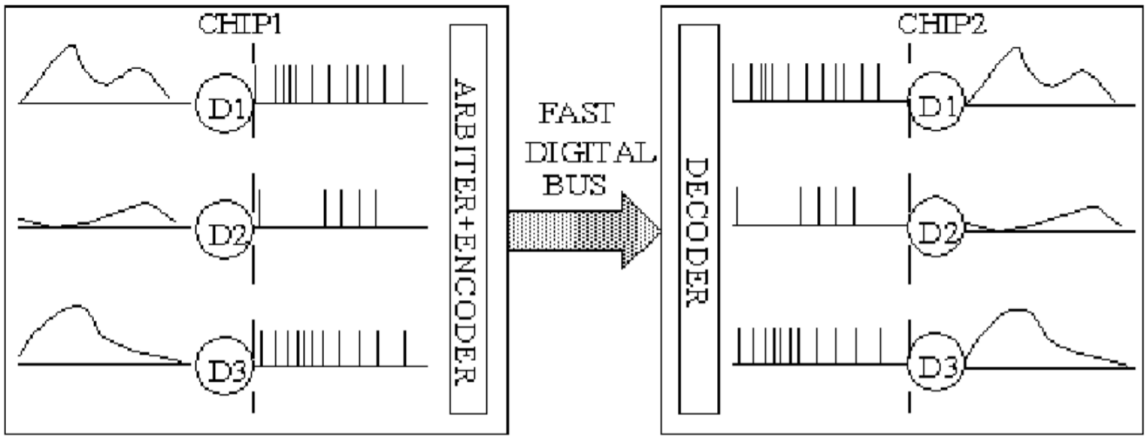


Figure 1.1: AER inter-chip communication scheme [1]

are two inconveniences. Firstly, designers may find it is inconvenient to design the asynchronous circuits using existing tools since the lack of a pre-defined library. The standard library, provided by the chip or the intellectual property (IP) vendors, is regarded as the key to speed up the design phase of integrated circuits (ICs) and often referred to as the success factor for the rapid growth of integrated systems [21]. However, building a standard cell library for asynchronous components can be challenging because the different styles of implementing asynchronous circuits require specific component sets. Thus, characterizing the electrical behaviour of asynchronous sequential components by the designer is demanded.

The second inconvenience is, researchers may find it is inconvenient to simulate the performance results of the chip, simply because there is no ready-made electronic design automation (EDA) tool for asynchronous circuits. For instance, there are many interconnected loops in an asynchronous circuit, making analyzing the performance of the circuit different from that of the clocked one. Hence, a back-end design flow is required to analyze the performance of the asynchronous design.

1.2 Approach

As part of the Innatera[1] initiative, this dissertation focuses on developing library establishment strategy and back-end design flow specifically for asynchronous circuits used in building AER communication blocks. The key to approach this is to well understand the target circuit. Proper decomposition of the circuit components helps to reduce the characterization workload. And familiarity with the operating mechanisms of the communication circuits helps to build the proposed asynchronous design flow based on the existing digital flow.

A diagram of the approach to the asynchronous design flow is shown as Figure 1.2. First of all, the asynchronous cells are customized specifically for AER communication blocks. The electrical and geometrical behaviours of the components are described by

[1]Innatera Nanosystems is the company where the author interned.

the Liberty timing file (LIB) and Library Exchange Format (LEF) file. The logical behaviours are described by the Verilog file. The design is implemented through logic design approaches but with several changes. Three simulations will be done after synthesis, placement and routing, and SPICE verification, respectively.

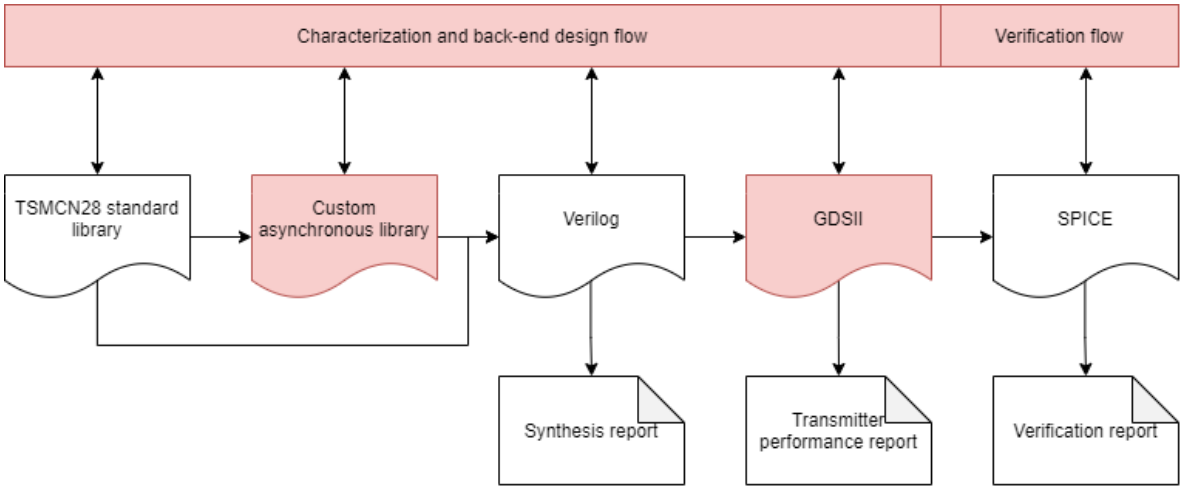


Figure 1.2: An overview of the asynchronous design flow

1.3 Goals

The main objective of this dissertation is to design a high-speed AER communication transmitter using the proposed library characterization and back-end design flow. To approach this main objective, several goals of the thesis are defined:

- Review the literature of sub-modules in AER transmitter design, and the literature of asynchronous design flow in commercial tools. Understand the operation mechanisms of the AER communication circuit.
- Define the asynchronous cells that are used in designing the AER transmitter block by circuit structural decomposition. Specify the logic, power arcs, timing arcs, etc. for characterizing the asynchronous cells at multiple processes, voltages, and temperatures (PVTs).
- Develop an asynchronous cell characterization and back-end design flow for the AER transmitter. Provide the solutions to the problems such as metastability, combinational loops, and interest timing path tracing, etc.
- Establish a verification flow to ensure the correctness and performance of the designed AER transmitter as well as the characterized asynchronous library.

1.4 Contributions

The main contributions of the thesis are:

- Provided an asynchronous cell library that supports the establishment of the AER communication circuit. The library contains the electrical, geometrical, and logical information of the cells.
- Provided a 32-bit AER transmitter based on the SR-latch implemented asynchronous components that are designed through the proposed back-end flow. The transmitter design is improved in the aspects of the arbiter architectures, encoder styles, and the "req_out" signal generation.
- Proposed a complete asynchronous circuit back-end design flow, which involves custom cell characterization, synthesis, post-synthesis verification, placement and routing, and static timing analysis. Also, provided a verification flow to validate the functionality of the resulting design from the back-end flow.

1.5 Outline

The dissertation is divided into three parts:
Part I: Literature Review

- Chapter 2 introduces the background for AER communication, discusses the state-of-the-art mechanisms and architectures of sub-modules in AER transmitter.
- Chapter 3 discusses current asynchronous circuits characterization methodology in commercial tools.

Part II: Proposed Transmitter Architecture Design

- Chapter 4 presents the AER transmitter architectures implemented with several to be compared modules, and explain the circuit diagram, signal transition graph, and timing diagram in detail.

Part III: Results

- Chapter 5 discusses the proposed asynchronous circuits characterization and verification flow, why the metastability filter is required, and also how to check the metastability by Monte Carlo simulation. The results of the transmitter performance are also reported in this chapter.
- Chapter 6 concludes this dissertation and talks about future work.

Part I
Literature Review

2 AER Communication Circuits

As it has been stated in Chapter 1, the communication scheme of AER includes translating a sequence of pulses generated by a set of cells into an ordered sequence of addresses. The addresses are sent through a digital address bus to the periphery. The sequence of addresses is converted in the receiver through the AER address decoder into a sequence of pulses and transferred to their destinations.

Figure 2.1 details the functional block information of Figure 1.1. The pixel blocks represent neurons to generate events. Naturally, neurons are arranged in a 2D spatial grid in retina models. This is also popular when the number of neurons becomes large since the 2D array allows encoding the two-dimension addresses of the neuron that pulsed separately.

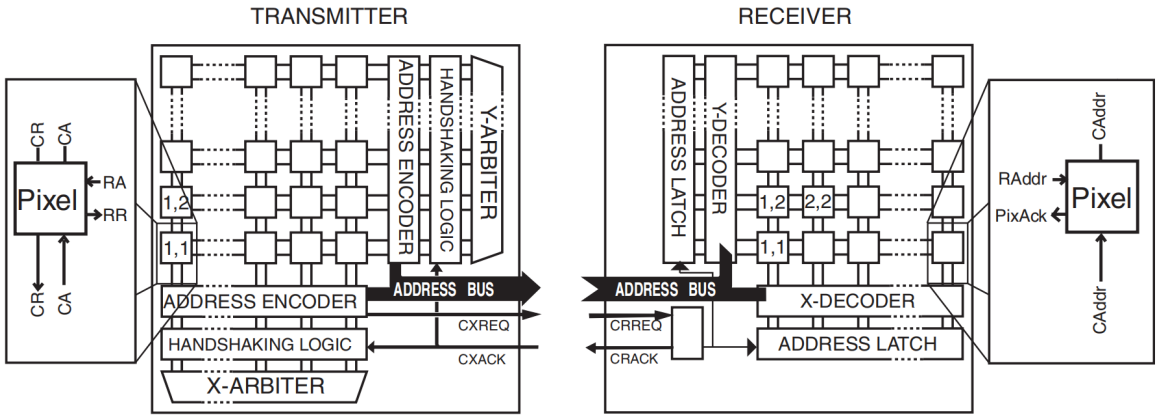


Figure 2.1: Block diagram of AER blocks on a 2D transmitter chip and 2D receiver chip [2]

This chapter aims at introducing various schemes of communication blocks in the literature for AER transmitters, which are different in their mechanisms for arbitrating and encoding. Different schemes can adapt to different types of neuromorphic systems because of their particular advantages and drawbacks.

2.1 AER Transmitter Blocks

Focusing on the transmitter as it is the target communication block to be designed, and also to verify the feasibility of the design flow, it has many individual axons from neurons as inputs. From this point, it is required to have the following three functions. The transmitter must first decide which spike to be communicated on the output channel next. This is related to the conventional arbitration problem. The address bus can assign a single address per time slot. If two or more spikes are produced at the same

time, the transmitter must determine in which order they are broadcasted through the address bus. To tackle such a simultaneous multi-event situation, an asynchronous arbiter is a solution. Secondly, since the address bus is shared by modules that operate separately and asynchronously, the AER bus access strategy must avoid bus collisions. In the third place, the transmitter must encode the address information of the axon and produce an output signal indicating the address generation after selecting the spiking axon. This is referred to as an encoding problem. N different spikes are encoded using $logN$ bits.

2.1.1 C-elements

The above mentioned three functions correspond to three modules in the design: arbiter, handshake module, and encoder, where the arbiter and the handshake module have handshaking protocol implemented. The C-element is used to create a handshake between two communication processes. C-elements are one of the most basic cells in asynchronous VLSI circuits. There are symmetric and asymmetric C-elements. Their symbols are plotted as Figure 2.2 and 2.3. The symmetric C-element is a state-holding element, with a high output when the inputs are high and a low output when the inputs are low. Any other input combinations have no effect on the C-element's state [22] (see truth table Table 2.1). Asymmetric C-element is an extended C-element allowing its inputs to only affect the element's operation when it transits in one direction. There are two types of asymmetric C-elements because of two directions (see truth tables Table 2.2 and 2.3 respectively).

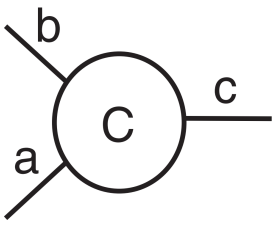


Figure 2.2: Symmetric C-element symbol

Table 2.1: Symmetric C-element truth table

a	**b**	$\mathbf{c_n}$
0	0	0
0	1	c_{n-1}
1	0	c_{n-1}
1	1	1

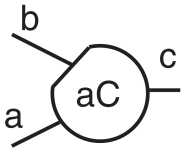


Figure 2.3: Asymmetric C-element symbol

Table 2.2: Asymmetric C-element I truth table

a	**b**	$\mathbf{c_n}$
0	0	c_{n-1}
0	1	1
1	0	0
1	1	1

Table 2.3: Asymmetric C-element II truth table

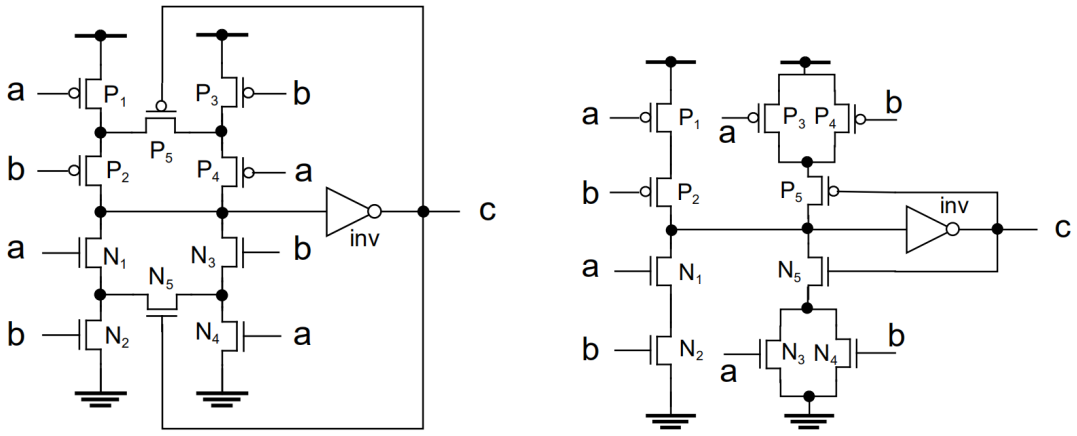
a	**b**	$\mathbf{c_n}$
0	0	0
0	1	1
1	0	0
1	1	c_{n-1}

The architectures of the C-element are various. For example, symmetric C-element can be implemented as in Figure 2.4a according to (2.1). As (2.1) can be rewritten to

(2.2), the CMOS implementation can also be changed to as Figure 2.4b.

$$c_n = a_n \times b_n + a_n \times c_{n-1} + b_n \times c_{n-1} \tag{2.1}$$

$$c_n = (a_n + b_n) \times c_{n-1} + a_n \times b_n \tag{2.2}$$



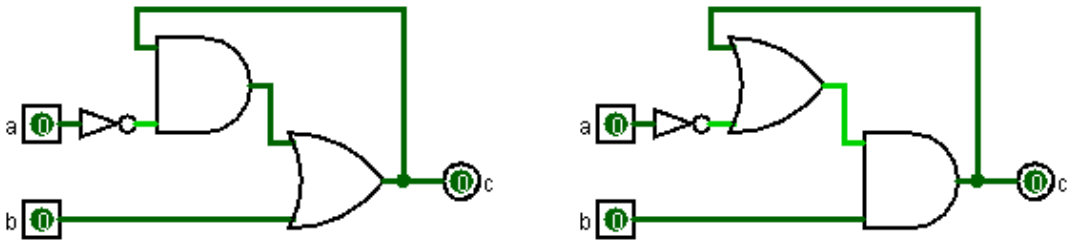
(a) (b)

Figure 2.4: Transistor implementation of symmetric C-element, based on (a) (2.1), (b) (2.2) [3]

Designing C-element from the transistor level is a full-custom asynchronous design approach. It sacrifices time to obtain well-sized C-element gates and thus better compatibility with the design. C-element can also be built from the logic gate level that speeds up the phase of transistor sizing. For instance, as truth tables of the two asymmetric C-elements can be written as (2.3) and (2.4), the logic gate implementations of asymmetric C-elements are presented as Figure 2.5a and 2.5b.

$$c_n = c_{n-1} \times \overline{a_n} + b_n \tag{2.3}$$

$$c_n = (c_{n-1} + \overline{a_n}) \times b_n \tag{2.4}$$



(a) (b)

Figure 2.5: Logic gate implementation of asymmetric C-element (a) I, (b) II

2.2 Arbitration Mechanisms

Arbitration is the process of selecting the access order to a shared resource among asynchronous requests [2]. There are several different architectures of arbiter according to different arbitration mechanisms. In the AER transmitter, the arbiter occupies most of the area and delay. Thus, the selection of arbiter architecture is important to the performance of the transmitter.

The key to ensuring that communications along several input channels are mutually exclusive in the arbiter is the mutual exclusion element (MUTEX). It is a two-input device guaranteeing the early coming request will be acknowledged first, and the inputs are mutually exclusive. The circuit of MUTEX comprises a NAND SR-latch and a metastability filter (MSF). The latch provides two stable states separated by a metastability state when the two inputs are both high. The filter is for restoring the metastability to a certain voltage level. It can be implemented by using transistors or logic gates as Figure 2.6a and 2.6b shown, respectively. The logic gate implementation consumes more area but achieves HDL synthesizability.

Two-way arbiters, such as two-way MUTEXes, can be used to make N-way arbiters, for example, constructing traditional arbiters like mesh, tree, and token ring arbiters. In the next sections, the mechanisms and architectures of those arbiters will be introduced.

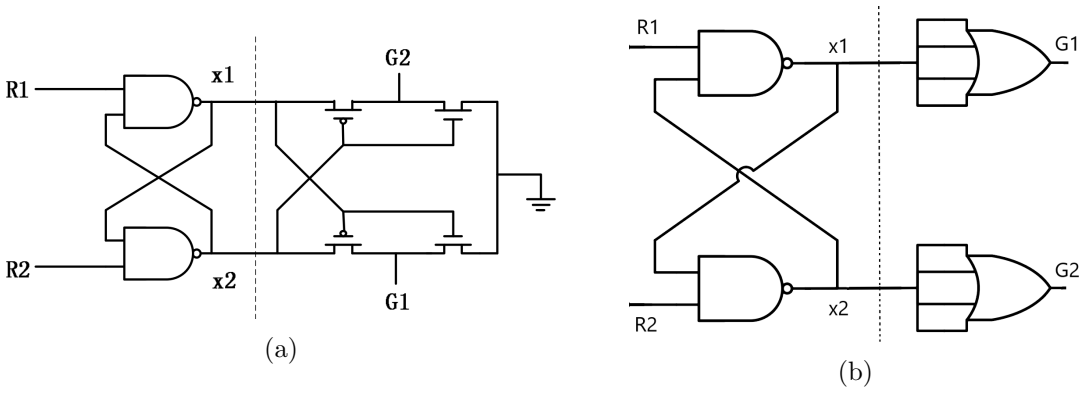


Figure 2.6: MUTEX cell implemented with (a) transistor based MSF [4], (b) logic gate based MSF [5]

2.2.1 Mesh arbiter

An n-way mesh arbiter has a basic structure that can be realized by arbitrating combinations on the 2-of-n basis [4]. Figure 2.7 shows a four-way mesh arbiter consisted of six MUTEX cells. The arbitration mechanism of the mesh is to compare the incoming order of the request one by one. For instance, G1 will win the arbitration if R1 wins R2 (as MUTEX in the first comparison stage[1]), and R1 wins R3 (second stage) and R1 wins R4 (third stage). For n inputs arbitration, we would need C_n^2 MUTEX cells. Each request will propagate through $n-1$ stages to grant. It can be found that the mesh architecture's complexity increases quadratically with the number of inputs n,

[1]We call MUTEXes in the first column as first comparison stage, etc.

while latency is proportional to n as well [23]. As a result, when an arbiter has a large number of inputs, this architecture is impractical.

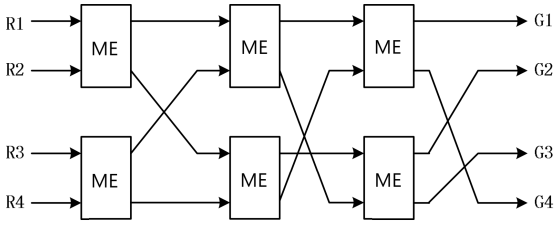


Figure 2.7: Four-way mesh arbiter [4]

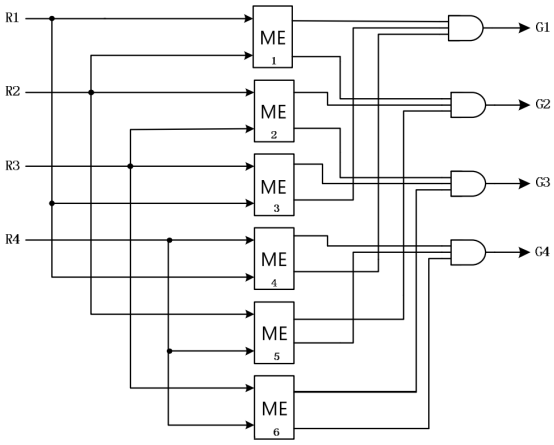


Figure 2.8: Four-way ordered arbiter [4]

The mesh arbiter architecture, because of its quadratically increased area consumption, has little literature focusing on improving its performance. Yu Liu *et al.* proposed an ordered arbiter working based on the mesh arbitration mechanisms [4]. It aggressively places the comparison logic all at the first comparison stage as Figure 2.8. That is to say, the comparisons of R1 vs. R2, R1 vs. R3, and R1 vs. R4 are all done at the first stage so that the latency remains minimal. The proposed ordered arbiter has the best latency comparing with other arbiter architectures [4], but its area consumption is even worse than the mesh arbiter because of the AND gates at the output.

2.2.2 Tree arbiter

The N-way arbiter can also be built using cascaded tree topology with $n-2$ tree arbiter modules and one MUTEX cell, as shown in Figure 2.9. The tree arbiter module has various kinds of implementation. Figure 2.10 shows one kind of implementation for the tree arbiter module. It is constructed by MUTEX cell, symmetric C-elements, and logic gates. There are two input channels and one output channel, each channel contains a request line and a grant line. The operation of the tree arbiter module is based on four-phase handshake protocol, which works either in the order IREQ1+ $\rightarrow$ OREQ+ $\rightarrow$ OGR+ $\rightarrow$ IGR1+ $\rightarrow$ IREQ1- or in the order IREQ2+ $\rightarrow$ OREQ+ $\rightarrow$ OGR+ $\rightarrow$ IGR2+ $\rightarrow$ IREQ2-.

In the cascaded tree arbiter, the signal OREQ in each tree module represents the input signals IREQ1 and IREQ2. It will propagate to the next level of the tree to be arbitrated with their neighbour cell in the same way. When the signal reaches the root of the tree, which is a MUTEX cell, the arbitration is resolved and the result will propagate back through the grant path. The request signal will pass through the $log_2N - 1$ tree arbiter modules forward to the root MUTEX cell, and then pass back through the same $log_2N - 1$ modules.

As is mentioned, the tree arbiter module has various kinds of implementation. A

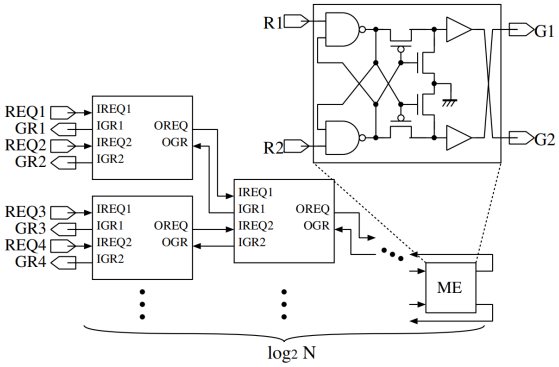


Figure 2.9: N-way tree arbiter [6]

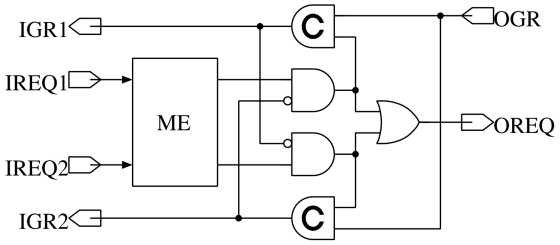


Figure 2.10: One of the schematic implementation for tree arbiter module [6]

comparison of different architectures of tree arbiter module is done by G. A. Subbarao *et al.* [5]. Under TSMC 65nm CMOS technology, the paper compares five novel tree arbiter module architectures with another architecture itself proposed, as shown in Table 2.4. The architectures include: a simple and popular arbiter that was proposed by J. Sparsø *et al.* [24]; a small in size but not HDL synthesizable arbiter that was proposed by K. A. Boahen *et al.* [25]; an arbiter with pipelined requests aiming at improving the response time that was proposed by S. R. Naqvi *et al.* [26]; a two-way arbiter improved in its concurrency based on the design of A. Ghiribaldi *et al.* [27] that was proposed by G. Miorandi *et al.* [28]; a unique arbiter containing no MUTEX but only C-elements which suffers from temporary deactivation of OREQ that was proposed by T. Turko *et al.* [29]; and an arbiter merging the C-elements into the logic with an HDL synthesizable MSF that was proposed by the author, G. A. Subbarao *et al.*

Table 2.4: Arbiter feature comparison [5]

		[5]	[24]	[25]	[26]	[28]	[29]
Num of transistors		62	74	44	116	82	70
Four-way handshake		Yes	Yes	Yes	No	Yes	No
Use MUTEX		Yes	Yes	Yes	Yes	Yes	No
Gate MSF		Yes	Yes	No	Yes	Yes	-
Average response time	**IREQ1↑-OREQ↑**	53.68ps	206.18ps	-	50.81ps	74.80ps	143.05ps
	IREQ1↑-IGR1↑	187.62ps	249.48ps	-	295.57ps	202.75ps	114.74ps

To conclude, Subbarao's arbiter wins in its well-balanced response time and number of transistors. It also supports HDL synthesis and four-way handshaking. For large scale Network-on-Chip, the number of transistors and the HDL synthesizability are two vital factors for the designer. Thus, Subbarao's arbiter is selected to be reproduced in our technology to compare its performance with the arbiter implemented using our proposed designing strategy. The comparison result is shown in Section 5.6.

2.2.3 Token ring arbiter

A token ring arbiter consists of multiple nodes, each of which is linked to a module that sends out requests through a channel. A token rotates around the ring indefinitely,

visiting nodes and polling requests in topological order. The node to which the module is attached will obtain the token and thereby win the arbitration [4].

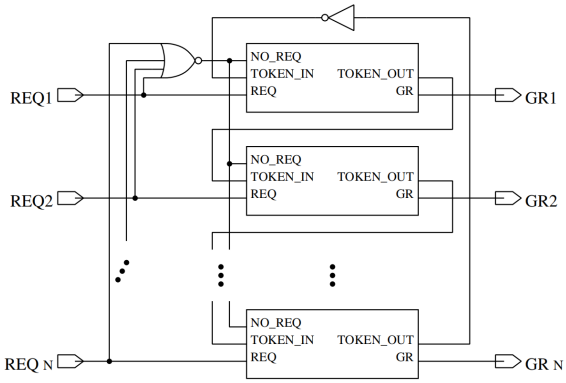


Figure 2.11: N-way ring arbiter [6]

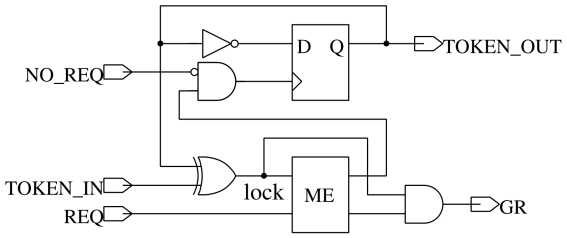


Figure 2.12: Ring arbiter module [6]

There are two types of token ring arbiters: busy ring arbiters and lazy ring arbiters [23]. The busy ring protocol is suitable for a heavy workload environment due to its faster response time, but it has a high power consumption. The lazy ring arbiter is advantageous in the inconstantly rotating token. Only when there is a request being activated, the token will move around in the ring, which saves power. But this will cause a delay in the token loop.

2.2.4 Arbiter architecture comparison

Some circuit designs and their comparison have been proposed [4, 6]. Aiming at 32 and above bits arbiter, the features of three arbiter architectures for large N is presented as Table 2.5, where the throughput is defined as the reciprocal number of the average interval between grant signals. Concluded from simulation results of Masashi Imai *at el.* as Figure 2.13 [6], the mesh arbiter[2] performs no better than the other two arbiter architectures in terms of latency, throughput, and area. Its area, as plotted in Figure 2.13g, is the worst because of its quadratically increased complexity. And its long propagation path leads to bad latency and throughput.

Table 2.5: Features of three arbiter architectures for large N ($N \geq 6$) [6]

	tree	**ring**	**mesh**
Latency	good	bad	bad
Throughput	bad	good	bad
Area	good	good	bad
Energy	good	bad	good

Tree arbiter wins the comparison of average latency. Its worst latency is better than the ring but its best latency is worse than the ring. This is because, for the cascaded

[2]There are two kinds of mesh arbiters in the figure: tri-mesh and square mesh. The tri-mesh arbiter is an unfair arbiter and the square mesh arbiter is an improved fair arbiter. Because the two arbiters do not show their advantages in the simulation result, here we do not explain them in detail.

tree arbiter, the latency for each request path is averaged because of its symmetric structure. For ring arbiter, the latency for each iteration in the simulation will be different according to the token location. For example, the worst-case latency happens when the nth request is active, but the token is located in the $n+1$th ring arbiter module so that the token has to propagate through all the other $n-1$ ring arbiter module back to grant the nth request. Besides, the paper also reported that the area of the ring is slightly larger than that of the tree. And its average energy-per-cycle is also larger. But because of its uncertain timing performance in the best and the worst cases, it is also selected to be reproduced in our technology to see which architecture is better for this design.

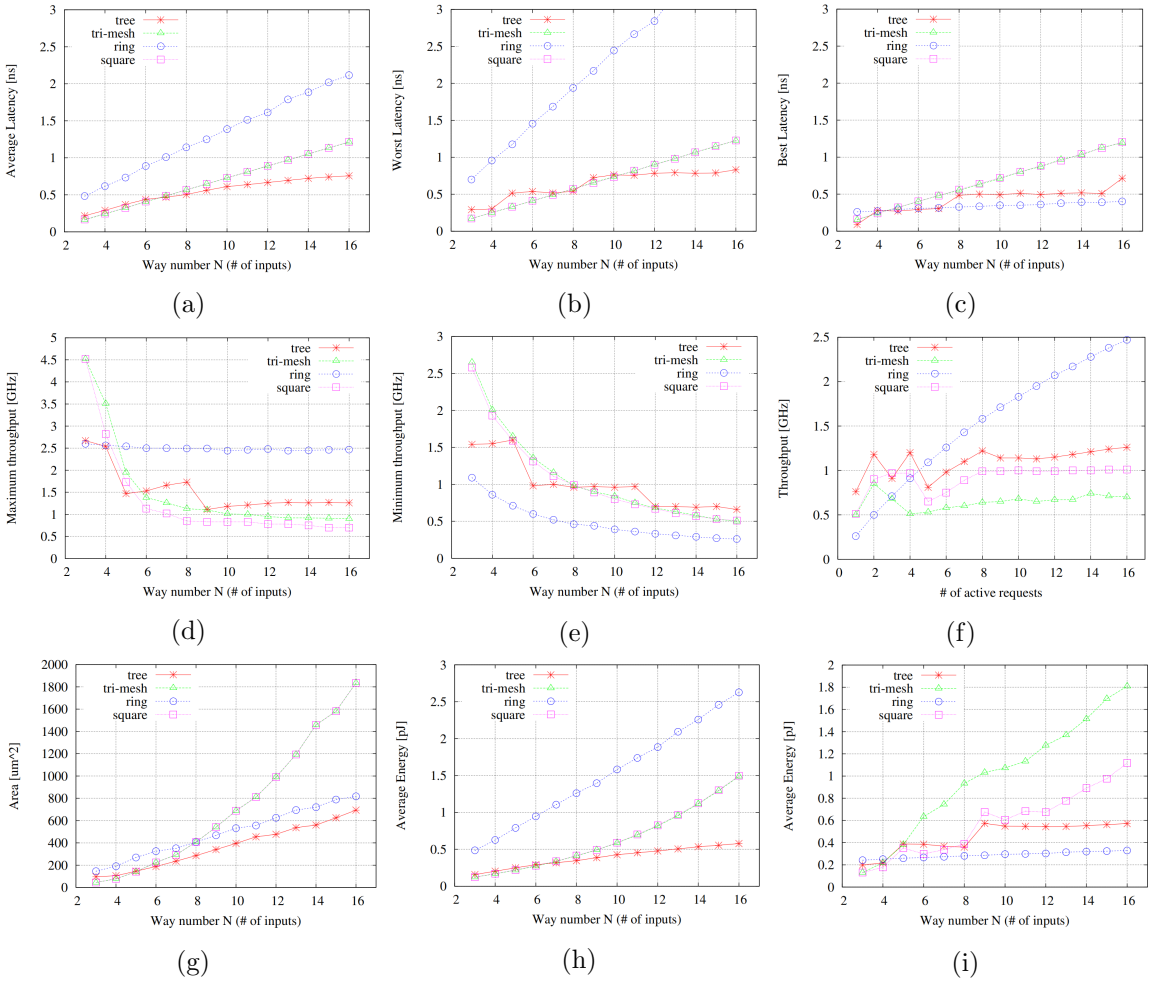
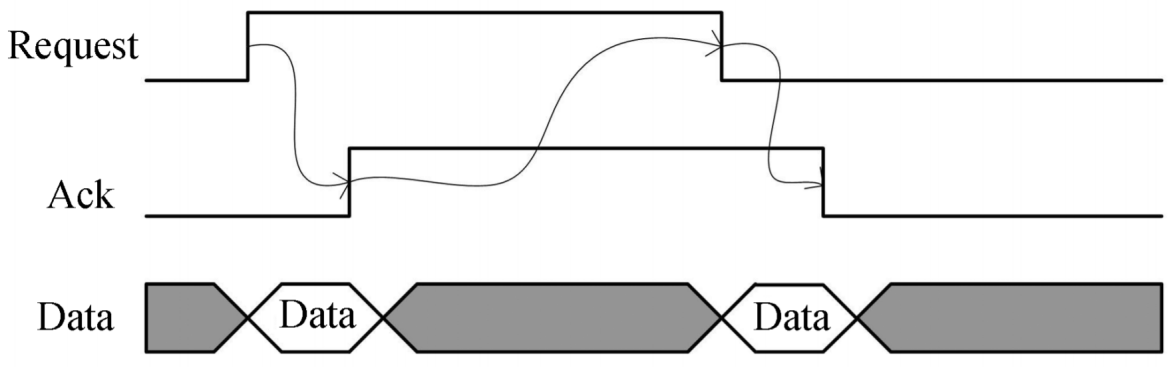


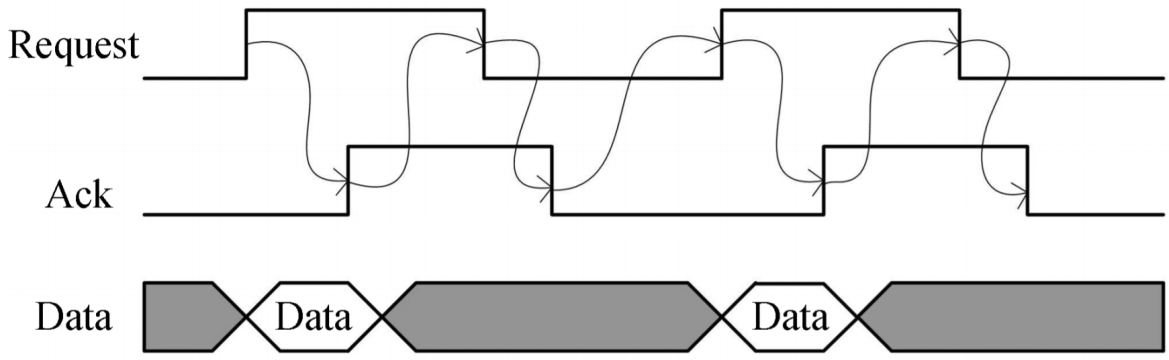
Figure 2.13: Comparison results of the mesh, tree, token ring arbiters: latency (a) in average, (b) of the worst path, (c) of the best path; throughput (d) in average, (e) of the worst path, (f) of the best path; (g) area, (h) average energy-per-cycle, (i) average energy-per-cycle under the maximum throughput condition [6]

2.3 Handshake Mechanisms

In the transmitter design, the acknowledgement signal generated by the arbiter will be encoded to address and passed to the receiver. Hence, the operation of the arbiter should be stable and synchronized to the external handshake. Figure 2.14a and 2.14b show the signal transition diagrams of two handshake protocols. The two-phase operation mode, benefited from that it can work at both the rising and falling edge of the request signal, has twice throughput as much as that of four-phase protocol circuits under the same frequency. But also because of this, the circuit is required to be edge-sensitive and has high complexity. The four-phase protocol does not suffer from such a transition problem. Only rising (falling) transition events or signal levels will be considered. Despite slow in comparison with the two-phase mode, the four-phase protocol is robust and can be accomplished with conventional logic circuits. As a result, most modern self-timed circuits use a four-phase handshake as their preferred implementation method [30].



(a)



(b)

Figure 2.14: Handshake transmission protocols: (a) two-phase handshake protocol, (b) four-phase handshake protocol [7]

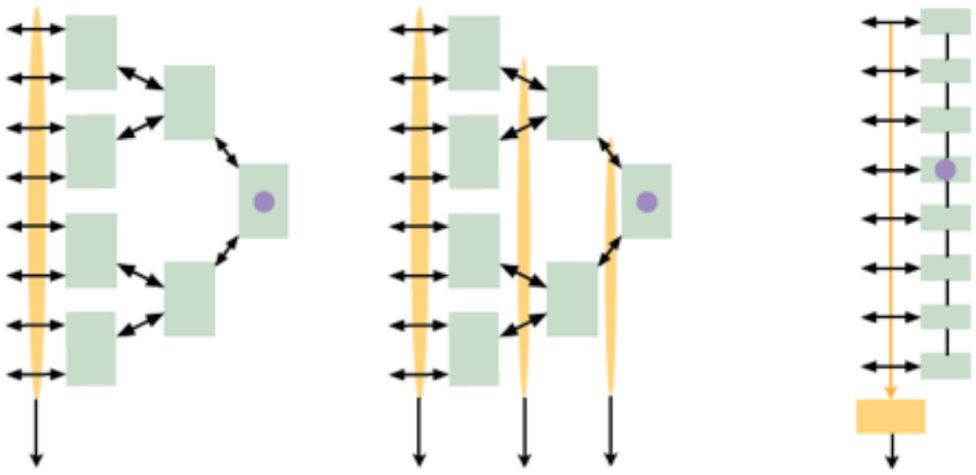
The transmitter communicates using a four-phase handshake protocol. To hold the four-phase handshake cycle until the address event is sent, an external handshake block is required. It latches the request and grant signals between the spike inputs and the

arbiter such that they are synchronized with the external acknowledgement "eack". To be specific, the block guarantees that the request into the arbiter stays high until it has been granted by both the arbiter and the external receiver, and a new arbitration is latched and postponed until the external acknowledgement has gone low again before starting a new cycle.

2.4 Encoding Mechanisms

The address encoder encodes the one-hot grant signals generated by the arbiter. For the 32-bit AER transmitter, the address encoder is a 32 to 5 encoder. Except for address output, there should be a signal called "req_out", which indicates that the address is ready to be read by the receiver. When the "req_out" is high, the receiver will read in the address and decode it to pulse its corresponding neuron cell. Hence, the "req_out" should be active only if all the addresses are available.

To reduce the encoder area specifically for tree arbiter, a hierarchical structure is applied instead of the basic encoder structure. For conventional logarithmic encoder as Figure 2.15a, each acknowledgement signal line connects to $O(logN)$ transistors. Thus, the address encoder consumes $O(NlogN)$ transistors in total to generate the encoded address. The hierarchical encoder structure (Figure 2.15b) allows each bit of the address to be encoded at each level of the tree. For instance, the root of the tree determines whether the most significant bit of the encoder output is 0 or 1, etc. By splitting the encoder levels, one cell in the arbiter connects to only two transistors. The number of the used transistor is reduced to $O(N)$. This technique cannot be applied to the ring arbiter because the ring arbiter does not have such a hierarchical structure as Figure 2.15c.



(a) (b) (c)

Figure 2.15: Encoder structures for arbitration: (a) basic tree encoder structure, (b) hierarchical tree encoder structure, (c) token ring encoder structure [8]

2.5 Combined Operation

To conclude, the target AER transmitter expanded from Figure 2.1 consists of three main parts: the external handshake modules, the arbiter, and the encoder. Two one-dimension transmitters can combine to achieve a two-dimension transmitter. Hesham Mostafa *et al.* [9] provide a simple and straightforward block diagram of the transmitter, as Figure 2.16. The presented transmitter has 64 channels, each channel contains a request signal and an acknowledgement signal. The "nChip_ack_i" in the figure is the external acknowledgement "eack" in our definition. It is connected to all 64 external handshake elements to hold the four-phase handshake cycle until the receiver requiring the next address event. The arbiter tree is used to determine the access order if multiple requests are sent at one time. The arbiter architecture in our design is not restricted to be the tree. As mentioned in Section 2.2.2 and 2.2.4, G. A. Subbarao *et al.*'s tree arbiter and Masashi Imai *et al.*'s lazy token ring arbiter have been proven to be better in performance comparing with other mentioned arbiter architectures. These two arbiter architectures will be compared in Chapter 4 with another popular tree architecture, where the latter is not listed in Table 2.4, but widely used in arbiter design as [2, 6, 9]. The arbitrated grant signal will propagate back to the external handshake block, then encoded to address by the encoder. The 64-bit OR gate is for generating the "req_out" signal. However, in Mostafa's design, it does not consider that the "req_out" should be available after the address is ready. Also, the one-hot encoder is a conventional logarithmic encoder. As a result, our proposed AER transmitter design will be improved in the aspects of the arbiter architectures, encoder styles, and the "req_out" signal generation to achieve better performance and correct operation.

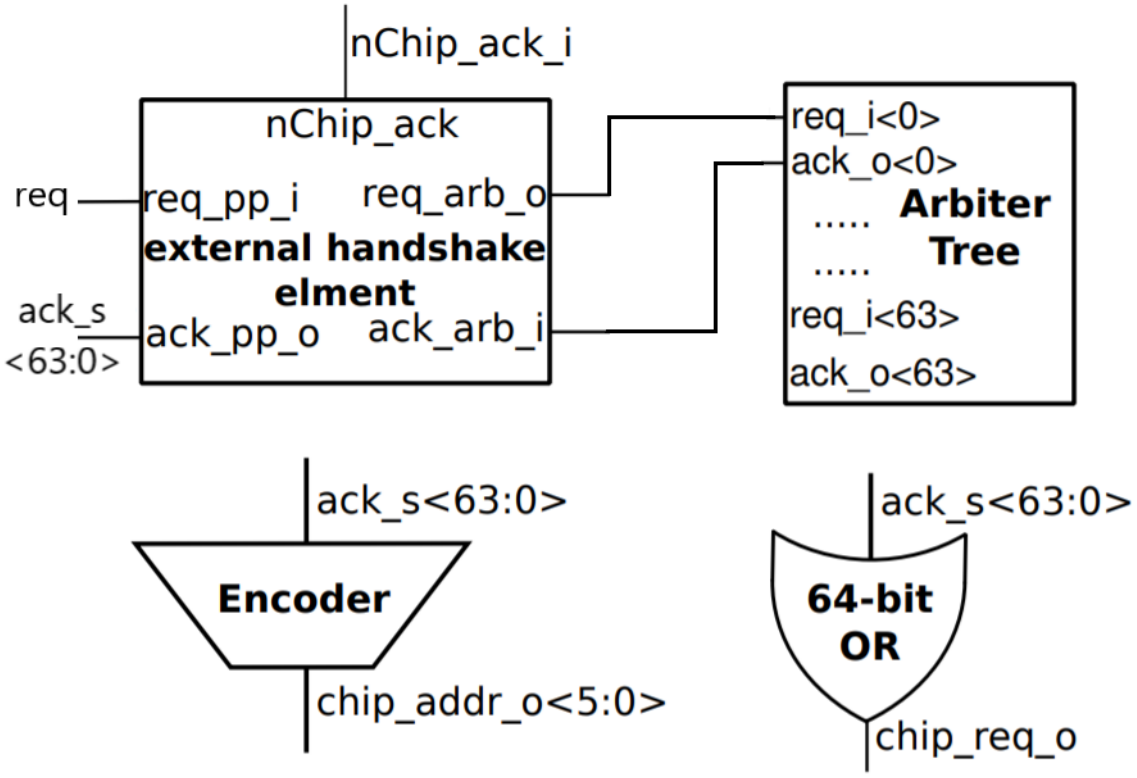


Figure 2.16: AER transmitter block diagram [9]

Asynchronous Circuits in Commercial Tools

3

With the development of clocked circuits and their computer-aided design (CAD) flow, the design difficulty of asynchronous circuits has been reduced compared to before. However, the challenge of universal adoption of this disruptive technology still exists. One of the crucial obstacles is how to apply the CAD flow of clocked circuits to asynchronous circuit design [31, 32]. Clocked CAD flows are more powerful than asynchronous ones, but they are incompatible to design sequential asynchronous circuits. Thus, there is widespread doubt about the ability to design commercial asynchronous circuits correctly and reliably.

Many research aiming at addressing the challenges work on the direction of integrating and adopting clocked CAD. Ad Peeters *et al.* [33] proposed an asynchronous design flow based on Tangram with the synchronous implementation of handshake circuits, which is advanced in technology and successful in commerce. In a different study field of desynchronization, Daniel H. Linder *et al.* [34] proposed a method of replacing each logic gate in a synchronous synthesized design with a small sequential handshaking asynchronous circuit. It is improved by Robert B. Reese *et al.* [35] to a coarse-grain approach to alleviate the excessive overhead caused by gate transformation. Current desynchronization approaches like [36, 37] are based on template and support Verilog, but they do not support general asynchronous design. Theseus Logic [38, 39] proposed a commercial tool flow based on clocked CAD that supports Verilog design descriptions. But this approach is only able to support quasi delay-insensitive null convention logic.

This chapter aims at stating the asynchronous circuit design flows proposed in literature, reporting the commercial tools used in the flows, and summarizing what changes the papers have made to the tools to make them suitable for asynchronous design. For some approaches in the proposed design flows, we learn from them and integrate them into our flow. Some methods and tools are abandoned due to reasons that will be explained in Chapter 5.

3.1 Asynchronous Cell Characterization

As is mentioned, the difficulty of establishing a standard cell library for asynchronous components comes from the different styles of implementing asynchronous circuits requiring specific component sets. Thus, different ways of characterization are proposed. This section mainly focuses on the case studies and analyzes their pros and cons if implemented in our flow.

- Jiang *et al.* [10] proposed a characterization method based on cutting feedback loops in asynchronous design. The target cell is a click element that has combinational loops inside. Its output pins function is more complex and indescribable to EDA tools. Thus, manually describe the pin functions of the click element, as

well as other standard asynchronous cells in their library, is impractical. Their method is to cut the loops when designing it in the library and reconnect the cut points when describing it in the Verilog. Albeit the characterization runtime is raised because of the increased number of pins, the method achieves characterization automation without using manual scripts. The click element is divided into a combinational logic part, buffers, and D flip-flop. And the layout is also arranged correspondingly so that every time when synthesizing, the output pulse width will not change too much according to the placement and routing. Such a customized cell achieves 20.8% less area and 63.7% less power consumption.
However, the paper did not mention anything about how to design and test a circuit system using the click element. Reconnecting the combinational loops will eventually cause trouble in static timing analysis because the current EDA tool is not able to analyze a circuit with complex loop feedback. If allowing tools to analyze it automatically, it will break the loop by inserting loop breakers and that may cut the interest timing path. This proposed method inspires us to do circuit decomposition when trying to create a standard cell library for asynchronous design. Instead of cutting feedback loops but still defining combinational and sequential parts in one cell, we attempt to separate the whole system into a combinational part and a sequential part. The asynchronous cells belonging to the sequential part will need to be characterized. The paper also reminds us to first analyze whether the decomposition of circuits causes any trouble in timing analysis.

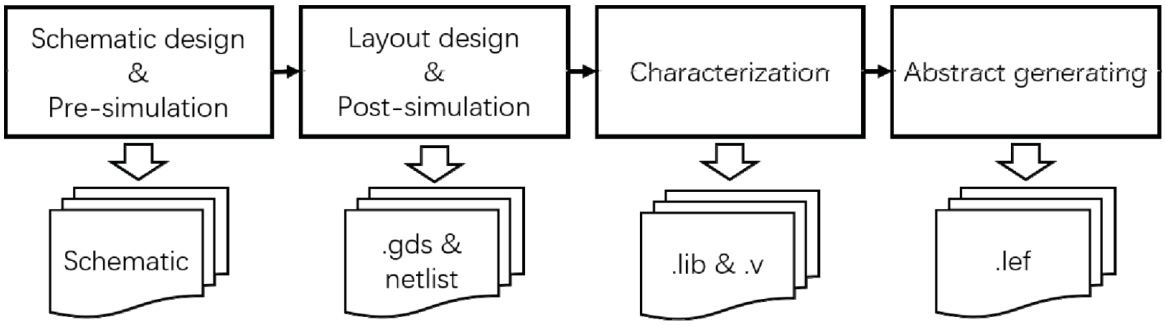


Figure 3.1: The design flow of standard cell libraries, proposed by Jiang [10]

- Moreira *et al.* [11, 40] proposed a Library Characterization Environment (LiChEn), which is an open-source electrical characterization tool for asynchronous standard cells. It is implemented using the C/C++ language and its characterization process is based on the generation of a SPICE simulation environment. The precision of the tool is verified by comparing seven logic gates' characterization results generated by the LiChEn and provided by the chip manufacturer, where the gates are from the STMicroelectronics 65nm CMOS technology library. And the performance of the program is verified through the time for characterizing 18 different types of C-elements in the ASCEnD library.
  Our specific target circuit, which is the AER transmitter, allows us to do circuit decomposition to reduce the complexity of the asynchronous cells so that the characterization workload and difficulty is lowered. An automated method is always

good, however, here we prefer to characterize asynchronous cells by using manually created scripts since it is not laborious for us. In the script, the timing and power arcs, as well as the logic of the cell, should be described. Moreira's work has a detailed explanation of how they defined those in the SPICE simulation environment, which is very helpful for our cell definition.

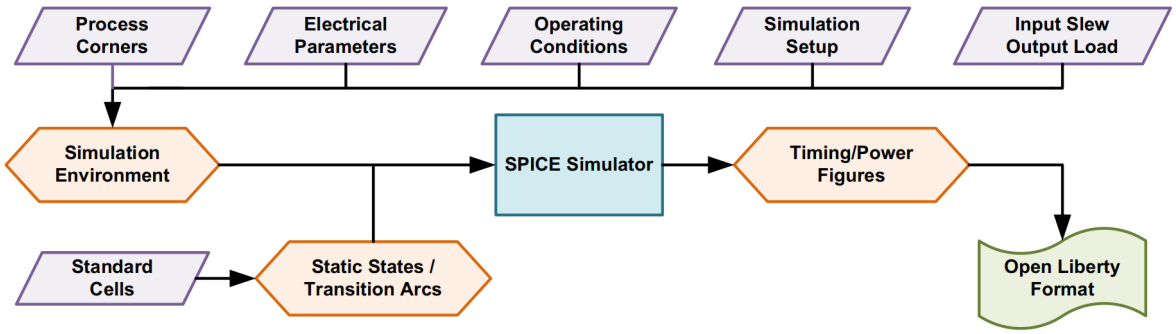


Figure 3.2: The LiChEn electrical characterization flow, proposed by Moreira [11]

- Cell characterization does not only include electrical information extraction, but also related to, for example, layout, symbol, and Verilog generation, etc. Moreira *et al.* [12] proposed the ASCEnD flow, a fully automated[1] flow for designing the components required for asynchronous systems using standard cells. The design flow has been used to establish a library of over 500 components based on the STMicroelectronics 65nm CMOS technology. And the components have been used to build three different network-on-chip routers and an RSA cryptographic core till layout level.
  The ASCEnD flow is complete in cell characterization. The LiChEn mentioned above is a part of the ASCEnD flow used to extract electrical characteristics of defined cells. This flow inspires us to develop our cell customization flow, which uses the tools of Cadence Virtuoso, Liberate, and Abstract Generator, to generate scs netlist, LIB, and LEF files respectively.

[1]The flow is automated except for the layout generation step.

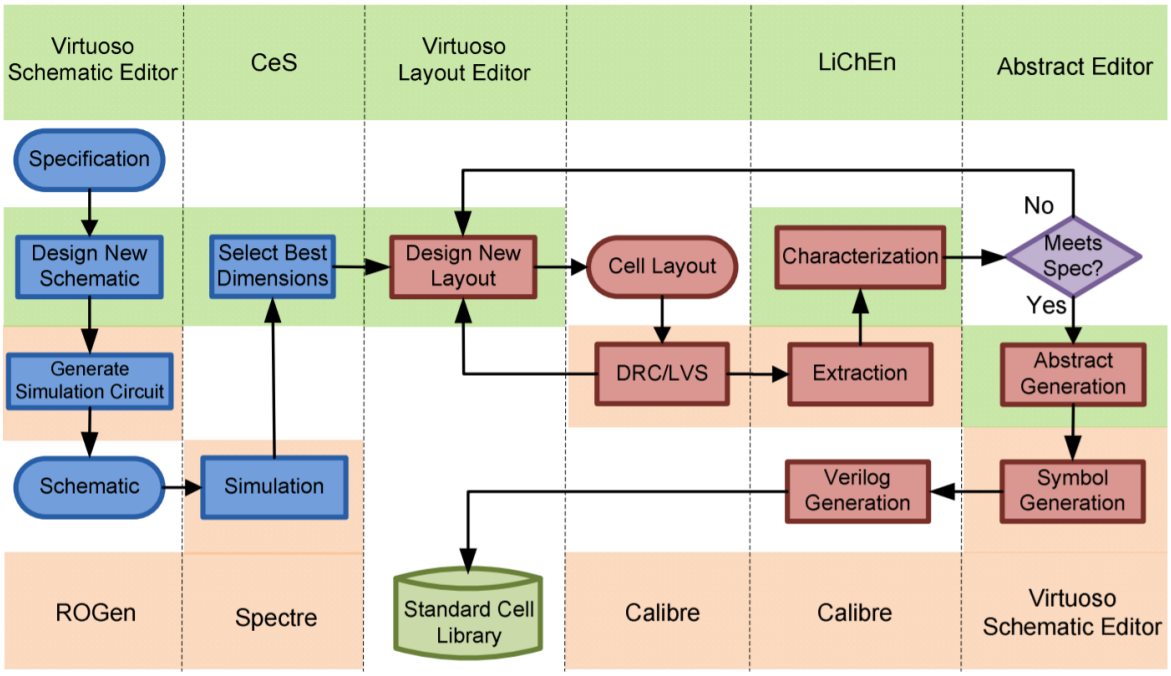


Figure 3.3: The ASCEnD design flow, proposed by Moreira [12]

Part II

Proposed Transmitter Architecture Design

4 AER Transmitter

Inspired by Mostafa's AER transmitter block diagram as mentioned in Section 2.5, the structure of a 4 channels demonstration of our AER transmitter is plotted as Figure 4.1. It will be improved in the aspects of arbiter architectures, encoder styles, and the "req_out" signal generation. An asynchronous back-end design flow has been established in Chapter 5 to evaluate the performance of the design.

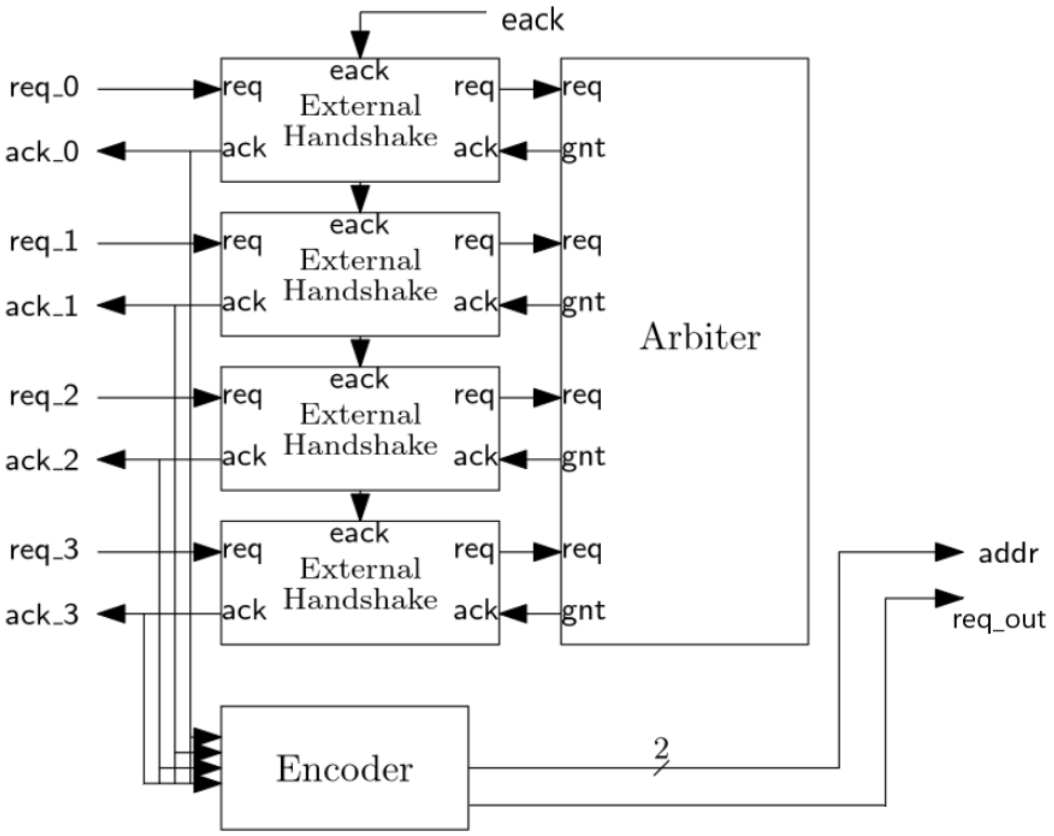


Figure 4.1: AER transmitter block diagram: a demonstration of 4 channels

In this chapter, we detail the circuit implementations of each component in the AER transmitter design. The different arbiter styles and architectures are illustrated. How the hierarchical encoder and external handshake are implemented is introduced. With this information, the circuit will be decomposed to find the needing cells used to build the transmitter but not included in the TSMC 28nm CMOS technology library, where the cells are required to be characterized.

4.1 Circuit Structural Decomposition and Cell Definition

Through studying the literature, there are three ways to design standard cells for asynchronous circuits. The first and also the simplest way is to use a behavioural model. The synthesis tool will read the behavioural HDL code description and find the corresponding asynchronous gates, for example, latch, in the standard library. This method benefits from that it does not require any custom library cell. However, cells in the standard library are generally built for synchronous design. Hence, this might not be an optimal solution in the power and timing performance aspects. Customizing cells at the transistor level is the optimal solution to create cells with the best performance. Designing at the transistor level allows designers to size the transistors to improve the design. But this method is time-consuming. It also requires many SPICE simulations to validate the design correctness. The third and preferred method is to customize cells at the logic gate level. It requires less SPICE simulation than customizing cells at the transistor level since the logic gates are provided by the foundry. Their functions and performances are verified by the foundry. Also, it provides better performance than the first method, since the cell can be customized by choosing logic gates with different driving strengths, where the latter differ in their area, power, and timing performance so that the proper ones can be picked to fit the design.

To design the asynchronous standard cell library at the logic gate level and obtain their electrical behaviours in characterization with less work, the target cells, as well as their schematics, should be defined. As described in Section 2.1.1, symmetric and asymmetric C-elements are needed to be characterized, where the former is used in the arbiter and the latter is used in the external handshake blocks. In Section 2.2, a MUTEX cell used in the arbitration is also introduced. They all contain a basic asynchronous gate structure inside, which is the SR-latch. SR-latch is the simplest bistable device. Information can be stored by the latch because of its feedback path. As a result, the latch functions as a memory device that can store one bit of data. The operation of SR-latches is independent of control signals and is solely dependent on the state of the set and reset inputs.

SR-latches can be constructed by two cross-feedback NOR gates or two cross-feedback NAND gates (usually called $\overline{SR}$-latch). The truth tables of NOR and NAND SR-latches are as Table 4.1 and 4.2, where the input combinations of "11" for NOR SR-latch and "00" for NAND SR-latch are two metastable states that do not allow to present. This is because practically one of the two logic gates will always win due to the manufacturing process, but there is no way to predict which it will be for a specific device from an assembly line.

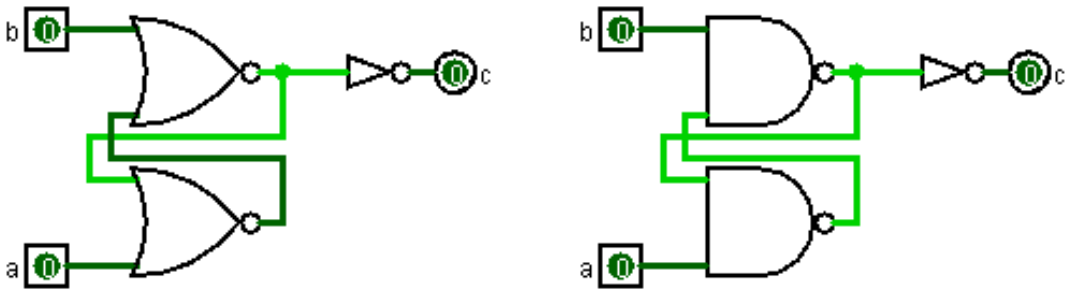
Table 4.1: NOR SR-latch truth table

S	**R**	$\mathbf{Q_n}$	$\overline{\mathbf{Q}}_\mathbf{n}$
0	0	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	0	1
1	0	1	0
1	1	Metastable	

Table 4.2: NAND SR-latch truth table

S	**R**	$\mathbf{Q_n}$	$\overline{\mathbf{Q}}_\mathbf{n}$
0	0	Metastable	
0	1	0	1
1	0	1	0
1	1	Q_{n-1}	$\overline{Q}_{n-1}$

By comparing the differences between truth tables of asymmetric C-elements (see Table 2.2 and 2.3) and SR-latches, it can be concluded that if making pin $\overline{Q}_n$ unconnected, the asymmetric C-elements can be constructed by simply placing an inverter at the pin Q_n of the SR-latches. To be specific, the asymmetric C-element I is built by a NOR SR-latch together with an inverter (as Figure 4.2a), and the asymmetric C-element II is built by a NAND SR-latch together with an inverter (as Figure 4.2b).



(a) (b)

Figure 4.2: Circuit schematics of the asymmetric C-elements, built by: (a) NOR SR-latch, (b) NAND SR-latch

The gate-level symmetric C-element can be implemented with a cross-feedback structure or a loop-back structure. The symmetric C-element in this design is constructed by a NAND SR-latch together with other logic gates (as Figure 4.3). Also, the MUTEX itself is a NAND SR-latch as Figure 2.6b. By decomposing the asynchronous circuits like so, in this design, there are only five cells having characterization necessity. They are NOR SR-latch, NAND SR-latch, NOR metastability filter, NAND metastability filter, and loop-break inverter. Where the latter three cells have the logic of an inverter but will be used in different places in the design. The metastability filter is used for cancelling the metastability of latches, this is described in Section 5.5. The loop-break inverter is for breaking the combinational loops in the asynchronous circuits, which will be introduced in Section 5.2.1. All the other modules in this design are built by the five cells and the standard cells in the TSMC 28nm CMOS technology library.

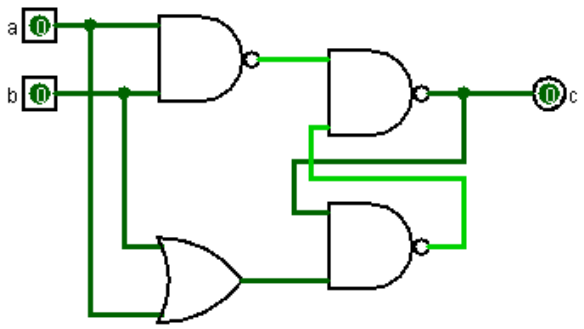
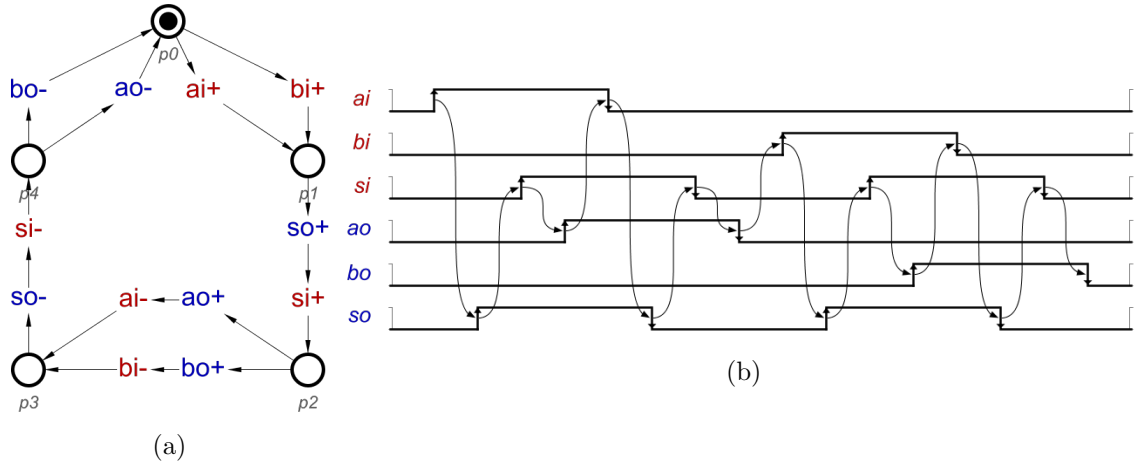


Figure 4.3: Circuit diagram of the symmetric C-element

4.2 Arbiter

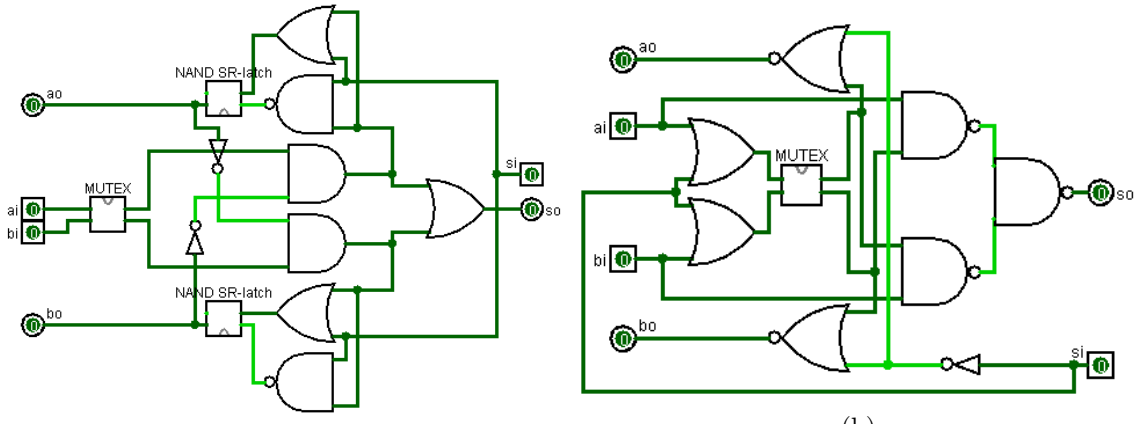
Figure 4.4a and 4.4b show the signal transition and timing graphs of the tree arbiter module. One of the two request signals, *ai* and *bi*, is selected by the MUTEX cell according to their incoming order to be granted. If *ai* is selected first, the acknowledgement of bi will be postponed until the finish of the *ai* grant, and vice versa.



(a) (b)

Figure 4.4: Tree arbiter module: (a) signal transition graph, (b) timing diagram

The two to be compared cascaded tree arbiter module architectures are introduced as Figure 4.5a and 4.5b. The first one is a traditional tree arbiter module that was not listed in Table 2.4's comparison. Here it is implemented with SR-latches in the MUTEX and symmetric C-element. The second architecture is the winner of the comparison, which was proposed by Subbarao *et al* [5]. The function of the C-element is merged into the logic gates and the loops in this design.



(a) (b)

Figure 4.5: Tree arbiter modules to be compared: (a) a widely-used and traditional architecture implemented with SR-latches, (b) architecture proposed by Subbarao *et al.* [5]

Another comparison candidate of the arbiter architecture is the token ring arbiter.

Its ring arbiter module is plotted as Figure 4.6. It is constructed by MUTEX, logic gates, and a sequential component: D flip-flop. But the D flip-flop here is used as a control logic to manage the token access instead of storing bit.

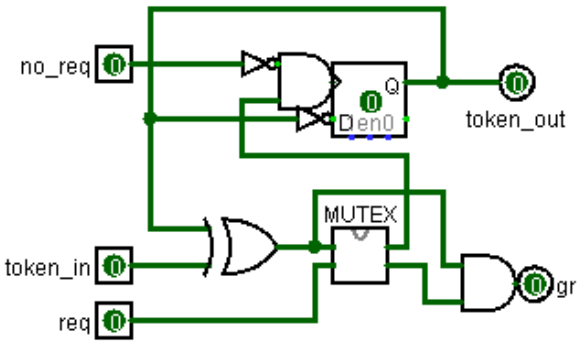


Figure 4.6: Token ring arbiter module

The three architectures are all implemented in the proposed asynchronous back-end design flow to compare their performances. The results of the comparison and the analysis are presented in Section 5.6. Briefly speaking, the traditional tree arbiter module with SR-latches wins the comparison if assuming the request signals occurring with equal probability. As a result, this architecture will be used in the transmitter in later design.

4.3 Handshake Module

Here shows the schematic of the external handshake module as Figure 4.7. It is constructed by connecting asymmetric C-elements I and II implemented by NOR and NAND SR-latches. An inverter has been added to the "eack" pin to make it active low. The "req_n" and "ack_n" are two pins connecting to external circuits. The "req_arb" and "ack_arb" are two pins connecting to the arbiter inside the AER transmitter.

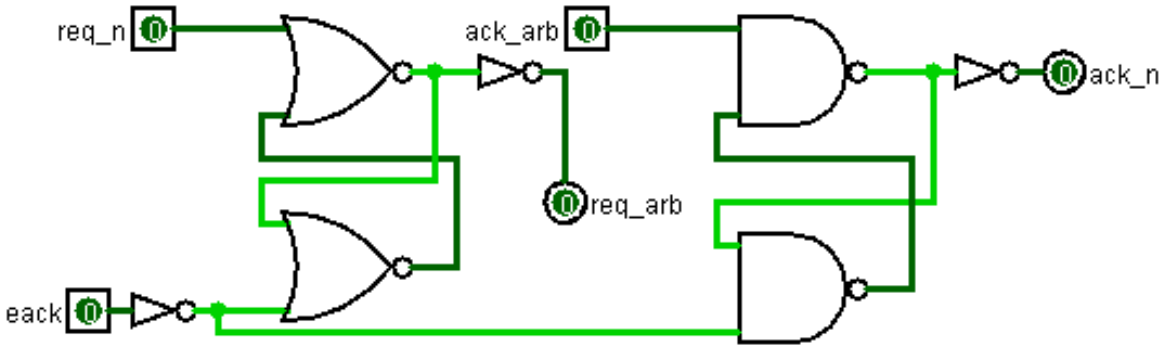


Figure 4.7: Circuit diagram of the external handshake module

There are two conditions to guarantee the correct operation of the AER transmitter: 1) The request into the arbiter should stay high until it has been granted by both

the arbiter and the external receiver. 2) A new arbitration should wait until the external acknowledgement has gone low again before starting a new cycle. With the two conditions being fulfilled, the operation of the external handshake is as follow:

1. All the inputs are set to be zero at the initial state.
2. When a request is received, the asymmetric C-elements I inside the external handshake module will send the signal to the arbiter. After arbitration, the corresponding grant signal will propagate from the arbiter to the asymmetric C-elements II.
3. When the external receiver is ready to read the address, the "eack" will turn high. Only when both "ack_arb" and "eack" are high, the asymmetric C-elements II can send the grant signal to the external receiver.
4. When the arbitration is done and its encoded address has been read by the external receiver, "req_n" and "eack" signals will turn low, resulting in the external handshake module back to its initial state. See Figure 4.8a and 4.8b for detailed operation.

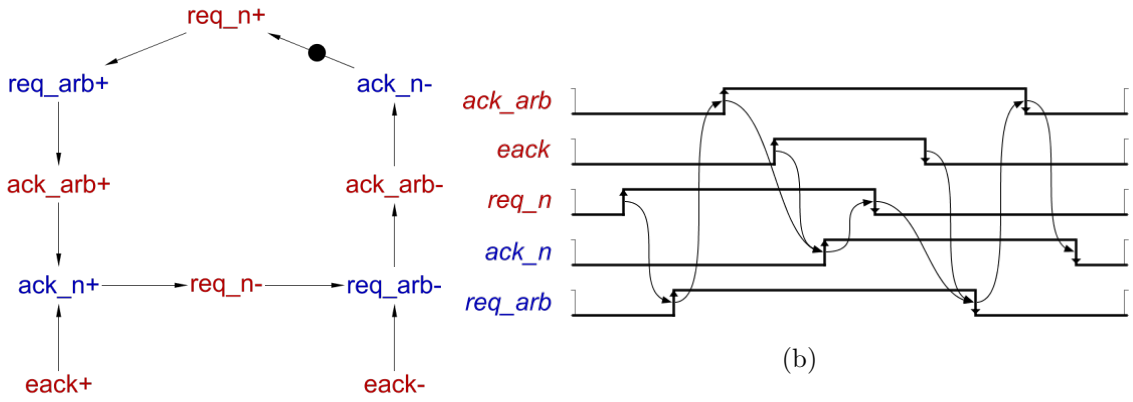


Figure 4.8: External handshake module: (a) signal transition graph, (b) timing diagram

4.4 Address Encoder

The address encoding part has been explained in Section 2.4. Here only introduce the method to generate the "req_out" signal following the rule of "req_out" must be ready after the address is available. At the very beginning phase of the design, this function is planned to be achieved by using a delay module. Listing 4.1 shows how it is implemented now with logic gates, where the "ack" is the acknowledgement signal generated by the external handshake module, the "ad" is the address encoded by the hierarchical encoder. The different addresses will have different logic to determine the generation of the "req_out" signal instead of postponing the signal by a constant time using the delay module. This achieves true asynchronous but costs more in area and power.

Listing 4.1: "req_out" generation Verilog code

```
always@(ack) begin
casez(ack)
32'h00000001: req_out = ack[0];
32'h00000002: req_out = ad[0];
32'h00000004: req_out = ad[1];
32'h00000008: req_out = ad[1] & ad[0];
32'h00000010: req_out = ad[2];
32'h00000020: req_out = ad[2] & ad[0];
32'h00000040: req_out = ad[1] & ad[2];
32'h00000080: req_out = ad[2] & ad[1] & ad[0];
32'h00000100: req_out = ad[3];
32'h00000200: req_out = ad[3] & ad[0];
32'h00000400: req_out = ad[3] & ad[1];
32'h00000800: req_out = ad[3] & ad[1] & ad[0];
32'h00001000: req_out = ad[3] & ad[2];
32'h00002000: req_out = ad[3] & ad[2] & ad[0];
32'h00004000: req_out = ad[3] & ad[2] & ad[1];
32'h00008000: req_out = ad[0] & ad[3] & ad[2] & ad[1];

32'h00010000: req_out = ad[4];
32'h00020000: req_out = ad[4] & ad[0];
32'h00040000: req_out = ad[4] & ad[1];
32'h00080000: req_out = ad[4] & ad[0] &ad[1];
32'h00100000: req_out = ad[4] & ad[2];
32'h00200000: req_out = ad[4] & ad[0] &ad[2];
32'h00400000: req_out = ad[4] & ad[2] &ad[1];
32'h00800000: req_out = ad[4] & ad[2] & ad[0] & ad[1];
32'h01000000: req_out = ad[4] & ad[3];
32'h02000000: req_out = ad[4] & ad[0] &ad[3];
32'h04000000: req_out = ad[4] & ad[3] & ad[1];
32'h08000000: req_out = ad[4] & ad[3] & ad[0] & ad[1];
32'h10000000: req_out = ad[4] & ad[3] & ad[2];
32'h20000000: req_out = ad[4] & ad[3] & ad[2] & ad[0];
32'h40000000: req_out = ad[4] & ad[3] & ad[2] & ad[1];
32'h80000000: req_out = &ad;
default: req_out = 1'b0;
endcase
end
```

4.5 32-bit AER Transmitter

One of the advantages of an asynchronous circuit is that it has high modularity. With the above-mentioned modules, the channels of the transmitter can be easily extended to higher bits by adding modules without changing the topology of the circuit. Similarly, the number of channels can be shrunk down to lower bits. Figure 4.9 presents the circuit diagram of an 8-bit AER transmitter as a demonstration to illustrate how the communication operates. The signal transition and timing diagrams of one request signal being activated are plotted as Figure 4.10a and 4.10b. The "ExHs" represents external handshake blocks. The "XARB" represents tree arbiter modules. The system starts at an all-zero initial state. Only when both request and external acknowledgement signals are high, the grant signal generated by the arbiter can be seen by the external system. The grant signal will be encoded to an address and then a "req_out" signal will be produced to indicate that the address is available to be read. Similarly, only when both request and external acknowledgement signals are low, the grant signal turns low, which causes address and "req_out" to change to low. When a request is chosen by the arbiter, all the other active request during the arbitration and transmission will be delayed till the beginning of the next cycle.

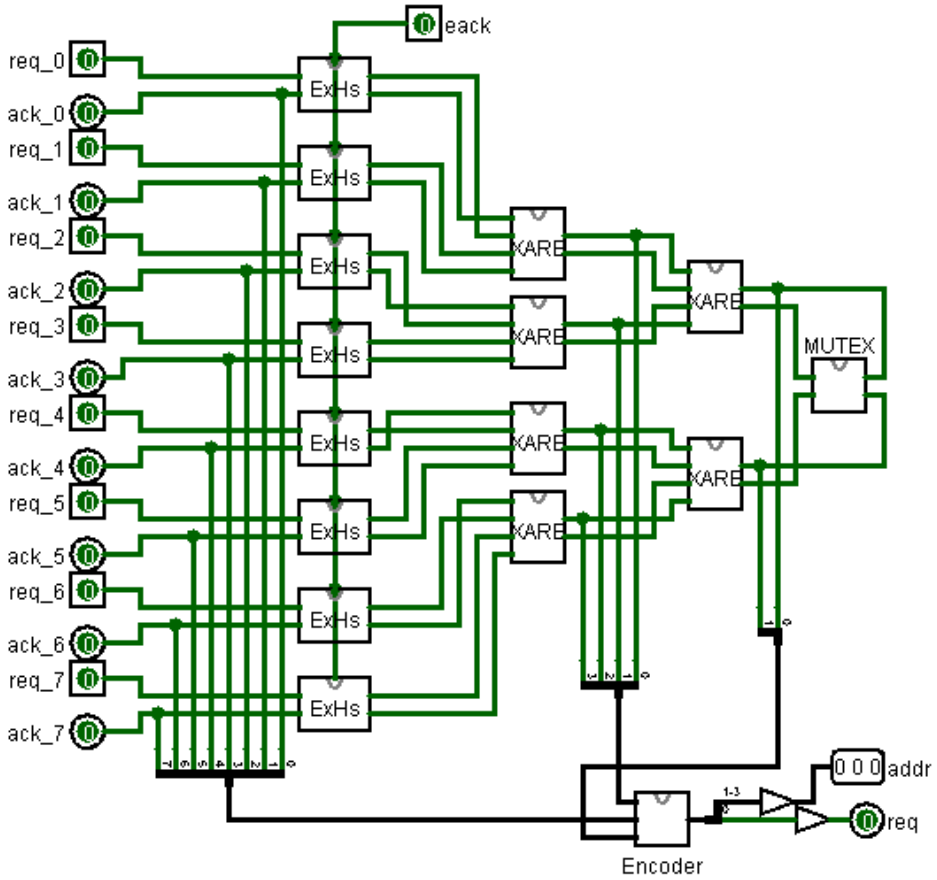
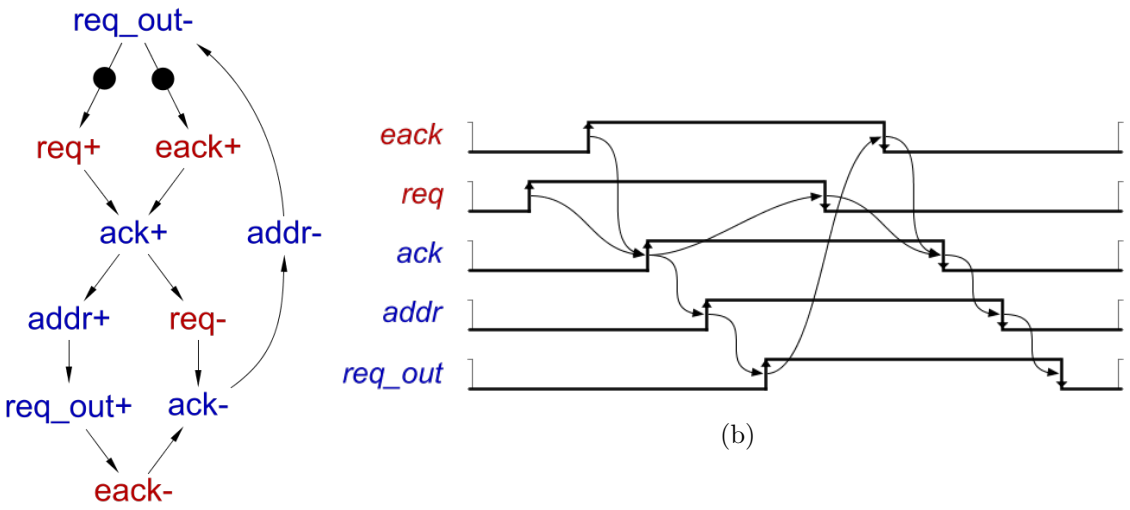


Figure 4.9: Circuit diagram of the AER transmitter: a demonstration of 8 channels



(a) (b)

Figure 4.10: AER transmitter: (a) signal transition graph, (b) timing diagram

Part III
Results

Proposed Back-end Design Flow, Verification Flow, and Results

5

The back-end design flow involves cell customization, synthesis, post-synthesis verification, placement and routing, and static timing analysis. The cell customization aims at generating the LIB and LEF files of asynchronous logic cells defined in this design. Those cells shall be constructed by standard cells and characterized for both timing and power so that the designers can design the asynchronous circuit just like designing a digital circuit without doing full-custom cell design and SPICE verification. The synthesis and static timing analysis are for checking the performance of the AER transmitter. There are combinational loops in asynchronous design, which shall be disabled to allow correct timing result generation. This chapter details the steps in the back-end design and verification flows, and presents the evaluation results of the AER transmitter.

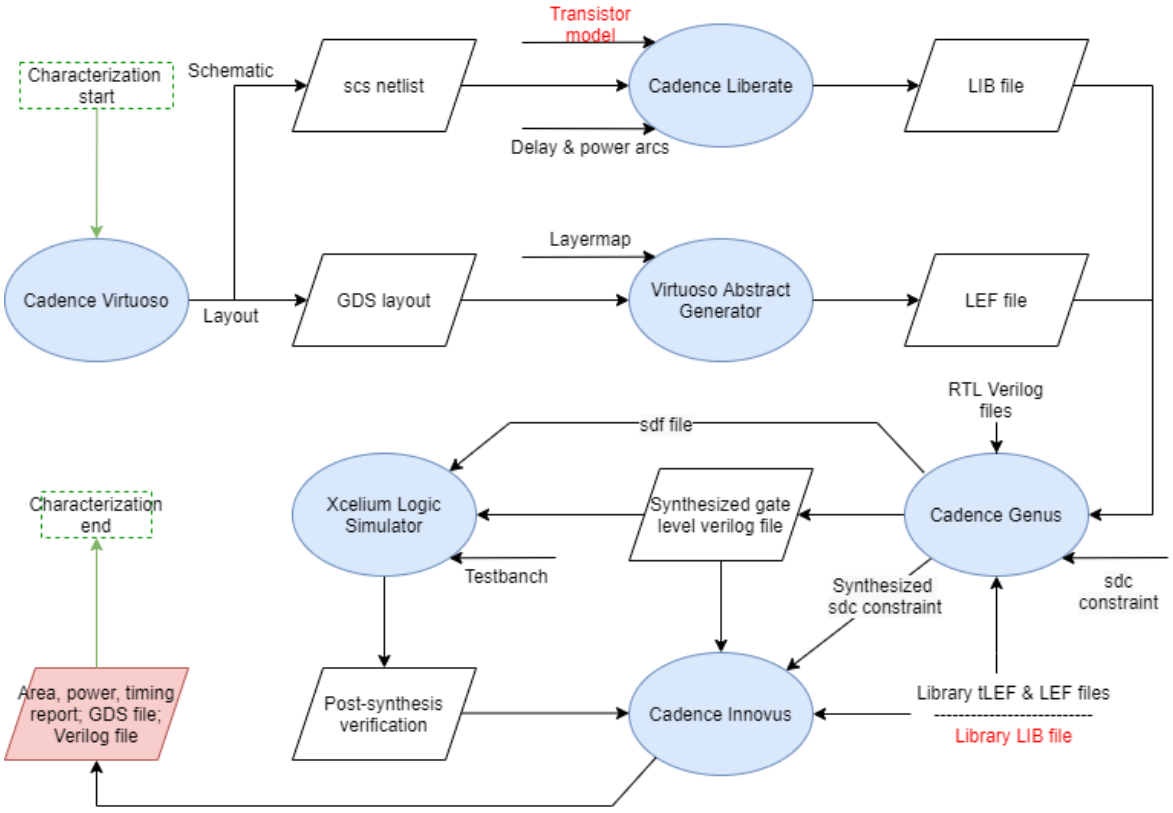


Figure 5.1: Back-end design flow chart

To be specific, the flow chart of the back-end design flow is shown as Figure 5.1. The ellipse represents the tools and the parallelogram represents the results or the resulting

files. The red labelled files shall be changed when changing PVT condition. The flow starts with needing cell customization. Five cells need to be characterized for this design. Their scs netlist and GDS layout will be generated by the Cadence Virtuoso. For asynchronous logic cells, delay and power arcs will be defined and given to the Liberate to generate the LIB file. After acquiring the LIB and LEF file of the custom cells, the Verilog file containing those cells will be synthesized and post-verified in Genus and Xcelium, respectively. Where in the synthesis step, the loop-break inverter will be inserted to the proper place to disable the combinational loops without breaking interest paths. After checking that the synthesis result is correct, needing files will be sent to Innovus to do placement and routing, and static timing analysis. A script will be used to track the interest signal propagation paths and generating the timing report for the AER transmitter. The design flow shall be able to adapt different PVT for supporting various PVT characterization. The flow ends with the generation of area, power, and timing reports; GDS file; and Verilog file of the AER transmitter.

5.1 Asynchronous Cell Characterization

Five cells have customization necessity for the AER transmitter, they are NOR SR-latch, NAND SR-latch, NOR metastability filter, NAND metastability filter, and loop-break inverter. They are constructed by the standard cells in the TSMC 28nm CMOS library, where the latches are used to build C-elements and MUTEX, the filters are used to cancel the metastability caused by the latches, and the loop-break inverters are used to break the combinational loops to correctly generate timing report. Cadence Liberate is used as the characterization tool to generate the LIB file of the five cells. It generates electrical cell views for timing, power, and signal integrity, including advanced current source models (CCS and ECSM).

Please see Section 5.5 for more explanation of how the metastability filters work and their schematics. The metastability filters and the loop-break inverter are acting as an inverter in the aspect of their digital logic. For Cadence Liberate, it can automatically generate the LIB file for such a one-input one-output device. For asynchronous cells like latches, their delay and power arcs shall be defined manually.

Typically for identifying timing arcs, pin to pin delays, as well as the corresponding output slopes, are characterized as a function of load and input slope. This enables slews to propagate during delay and timing analysis. In this design, a 5×5 template table is used, where the slew rate is defined to be 10ps, 25ps, 50ps, 75ps, and 100ps, and the load capacitance is defined to be 1fF, 5fF, 10fF, 15fF, and 20fF. For power, both dynamic and leakage powers are characterized. Dynamic power can be divided into internal power and switching power. The power dissipated by an instantaneous short-circuit connection between the supply voltage and the ground at the moment the gate switches state is known as internal power. Switching power is dissipated when the internal and net capacitance is charging or discharging, and leakage power is caused mainly when a transistor is switched off, undesired sub-threshold current flows through the transistor channel [41].

As a result, power arcs are defined in the characterization template as when input changes resulting in no output transitions. Timing arcs are defined as when input

changes causing output transitions. And the leakage power is defined as when the states in the truth table occur for each cell. For example, truth tables of NOR and NAND SR-latches are rewritten and presented as Table 5.1 and 5.2 considering metastability as a legal state. Each truth table contains five states because input combinations of "00" for NOR SR-latch and "11" for NAND SR-latch can be expanded to two states according to its different previous outputs.

Table 5.1: NOR SR-latch truth table

State	S	R	Q_n	$\overline{Q}_n$
①	0	0	0	1
②	0	0	1	0
③	0	1	0	1
④	1	0	1	0
⑤	1	1	0	0

Table 5.2: NAND SR-latch truth table

State	S	R	Q_n	$\overline{Q}_n$
①	0	0	1	1
②	0	1	0	1
③	1	0	1	0
④	1	1	1	0
⑤	1	1	0	1

Table 5.3 and 5.4 show the timing and power arcs definition of the NOR and NAND SR-latches. The circled number represents the state transition situation, and the 0, 1, and arrows represent the input or output transition situation. 0 or 1 means before and after the state transition, the logic of the particular pin remains unchanged at 0 or 1. The up arrow means the logic is transiting from 0 to 1, and for the down arrow, it represents the opposite.

Table 5.3: Timing and power arcs of NOR SR-latch

Delay arcs						Power arcs	
①④	↑ 0 ↑ ↓	③④	↑ ↓ ↑ ↓	⑤③	↓ 1 0 ↑	①③	0 ↑ 0 1
①⑤	↑ ↑ 0 ↓	③⑤	↑ 1 0 ↓	⑤④	1 ↓ ↑ 0	②④	↑ 0 1 0
②③	0 ↑ ↓ ↑	④③	↓ ↑ ↓ ↑	⑤②	↓ ↓ ↑ 0	③①	0 ↓ 0 1
②⑤	↑ ↑ ↓ 0	④⑤	1 ↑ ↓ 0			④②	↓ 0 1 0

Table 5.4: Timing and power arcs of NAND SR-latch

Delay arcs						Power arcs	
①②	0 ↑ ↓ 1	④①	↓ ↓ 1 ↑	③②	↓ ↑ ↓ ↑	②⑤	↑ 1 0 1
①③	↑ 0 1 ↓	⑤①	↓ ↓ ↑ 1	④②	↓ 1 ↓ ↑	③④	1 ↑ 1 0
②①	0 ↓ ↑ 1	①⑤	↑ ↑ ↓ 1	⑤③	1 ↓ ↑ ↓	④③	1 ↓ 1 0
②③	↑ ↓ ↑ ↓	③②	↓ 0 1 ↑			⑤②	↓ 1 0 1

In the arc definition, five arcs are not defined for each latch, namely for NOR SR-latch: ①②[1], ②①, ③②, ④①, and ⑤①; for NAND SR-latch: ④⑤, ⑤④, ②④, ③⑤, and ①④. They are absence because they are impossible to occur in the operation.

The characterization is done under different process, voltage, and temperature as Table 5.5 shown. The PVT conditions are chosen according to the existing TSMC 28nm

[1]This way of expression represents transiting from state 1 to state 2.

CMOS library. The LIB files after Cadence Liberate lacks cell area and some other configurations, where this information are written into another LIB file and merged with the resulting file using the "*merge_library*" command.

Table 5.5: PVT conditions

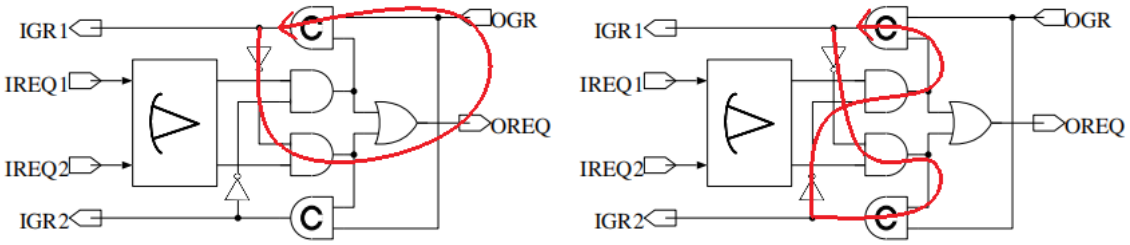
Corner	**Voltage (unit: V)**	**Temperature (unit: °C)**		
FF	0.88	0	-40	125
	0.99	0	-40	125
	1.05	0	-40	125
SS	0.9	0	-40	125
	0.72	0	-40	125
	0.81	0	-40	125
TT	0.8	25	85	
	0.9	25	85	
	1	25	85	

The LEF file of the five cells is generated by Cadence Abstract Generator. It takes the GDS layout as input, using the layermap file to generate the LEF file of the cells.

5.2 Synthesis

5.2.1 Breaking timing loops

The LIB and LEF files generated by cell customization helps the designer to design the asynchronous circuit as designing a digital circuit. The asynchronous cells are made as black boxes so that the loops inside the cell would not cause any problem in synthesis. However, the combinational loop does not only exists in the asynchronous cells but also in other modules of the design. A combinational feedback loop is a path that can be traced through combinational logic back to the starting point [42]. For instance, each grant path in the tree arbiter module has two combinational loops as Figure 5.2a and 5.2b. The loop is for arbitrating the signal considering the grant situation so that guaranteeing the correct operation of the tree arbiter.



(a) (b)

Figure 5.2: Grant path combinational loops in tree arbiter module: (a) I, (b) II

To analyze design with combinational loops, Genus requires to break the loop at

some point within the loop. By default, Genus recognizes each feedback loop, adds a buffer from the technology library on the path, and disables its timing arc from input to output, thus breaking the timing loop. These inserted buffers are easy to be identified since they follow the "*cdn_loop_breaker<number>*" nomenclature. However, the loop can be broken at multiple points. An improper breaking point in a loop might result in the interest paths being unintentionally eliminated in the timing analysis.

To avoid the problems caused by automatically breaking the loop, a loop-break inverter is customized and made available to be inserted to break the loop manually. The loop-break inverter has the same schematic and layout as an inverter in the TSMC 28nm CMOS library "INVD0BWP30P140". It is created to distinguish it from the original cell. It is written in the Verilog code to replace the inverter between the symmetric C-element and the AND gate in the tree arbiter module. And its timing arc is disabled in the synthesis constraint by using "*set_disable_timing [get_lib_cell loop_inv]*" command. The loop is broken before "*synthesize*" command, thus there is no need to remove the automatically inserted buffer by using the command "*remove_cdn_loop_breaker*" since there will be no loop detected and no buffer inserted.

The synthesis generates gate-level Verilog code of the design, Synopsys Design Constraints (SDC) file, and a Standard Delay Format (SDF) back-annotation file used in post-synthesis verification.

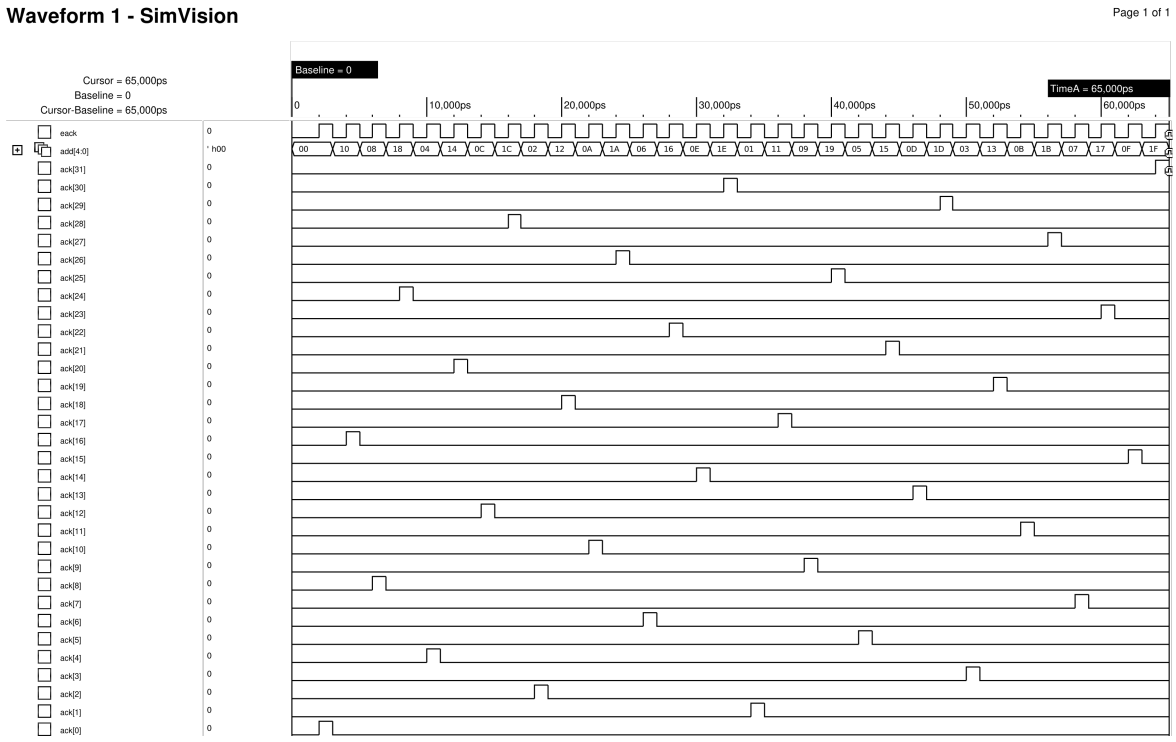
5.2.2 Post-synthesis verification

The post-synthesis verification uses Xcelium as a simulation tool. It ensures the logical correctness of the synthesized gate-level circuit and also allows back annotation to do a more realistic simulation.

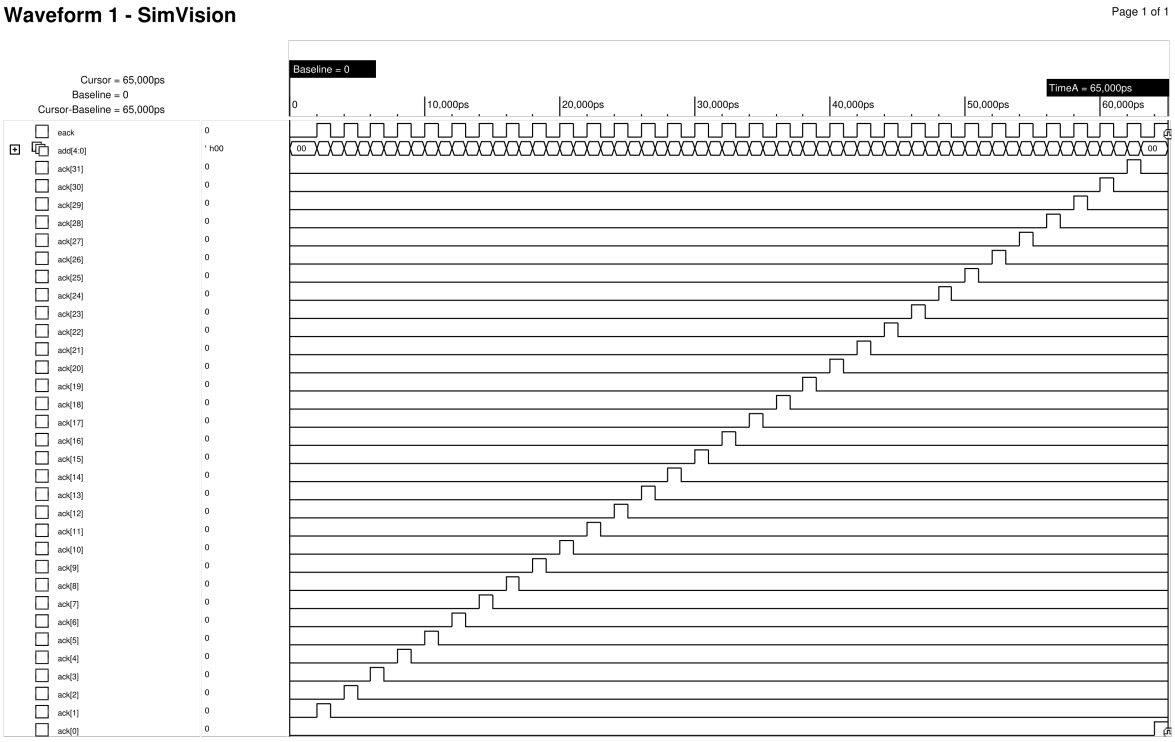
The testbench starts with an all one request, namely all request signals are being activated to see the response of the circuit. Figure 5.3a and 5.3b show the results of the simulation. The two AER transmitters after synthesis have been verified to have a correct function. From the results, it can be observed that the tree arbiter does not arbitrate in the topology order, but the ring arbiter does.

5.3 Static Timing Analysis

To check the performance of the AER transmitter, the area, power, and more importantly, timing information of the transmitter will be generated by Innovus. The Innovus working flow starts with loading files into the tool and floorplanning. Then the ports of the design will be assigned to physical pins along the perimeter of the design. The pins will spread along the left and right perimeters of the core area. The path groups will then be set up to prioritize which paths should spend more time fixing for the engine. The power ring will be added around the core, where two pins of VDD and GND will be created to fit the following GDS to SPICE conversion. After adding the well tap, the design will be placed and routed, and fillers will be added. The final resulting layout is shown as Figure 5.4. The address encoder located at the right side of the chip is constructed by a large number of logic gates. Thus, the wire connection density is large. Even though the arbiter in the middle of the chip has more gates utilized, but



(a)



(b)

Figure 5.3: AER transmitter post-synthesis verification result: (a) with tree arbiter, (b) with lazy token ring arbiter

gates are customized and characterized as standard cells as latches. So some of the wires are connected inside the blocks, which visually seems to have a lower density.

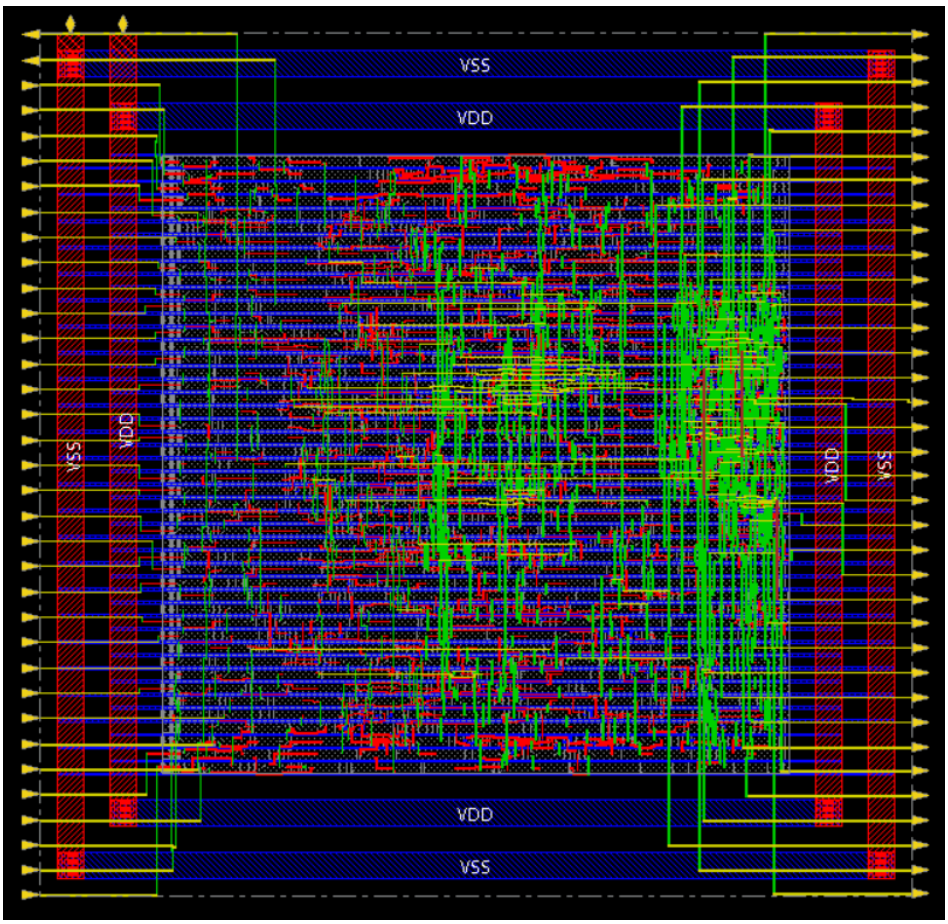


Figure 5.4: AER transmitter layout

To correctly report the interest timing information, it needs to be ensured that the engine is tracing the correct signal propagation paths. Because of the tree topology, the grant signal after arbitration has multiple paths to propagate back to the acknowledgement pins, in which the one that has the corresponding number with the request signal is the correct and interest path. The ring arbiter does not have such a problem, but it requires generating the token loop delay information. The token loop delay is measured by summing the delays of each loop sections. The loop section is defined to be the delay from the clock pin of the D flip-flop in the ring arbiter module to the next clock pin of the D flip-flop in the next module.

Table 5.8 shows an Innovus timing report example of the AER tree transmitter's worst delay. After the 31st request signal being activated, the external handshake module will send that signal to the tree arbiter. The signal will go through the corresponding tree arbiter modules in the four-level tree and reach the MUTEX at the root of the tree. After arbitration at the MUTEX, the grant signal will propagate through the grant path of the same four tree arbiter modules and back to the handshake module. The grant signal is passed from the arbiter to the external system according to the

"eack" signal. At the same time, the signal also enters into the address encoder. And finally, the addresses and "req_out" signal will be produced.

Table 5.6 lists the same case but for the AER ring transmitter. A timing report of one token loop section inside the ring arbiter is also included as Table 5.7. As mentioned in Section 5.6, the worst delay of the ring reported by Innovus is the worst delay of the best case, which does not consider the token loop. Thus, the request signal from the handshake module will directly pass through the ring arbiter module to become a grant signal without considering the token location. To evaluate the true worst delay of the worst case, a sum of token loop delays will be added to the above delay. As in the table, the loop section is measured from the clock pin to the next clock pin of the D flip-flop in neighbour ring arbiter modules.

Table 5.6: AER ring transmitter worst delay Innovus timing report

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	req[31] ^			0.000	0.412
external_hs[31].exhs_NRSR1	R ^ ->Q v	NOR_SR	0.006	0.006	0.418
external_hs[31].exhs_NDMSF1	I v ->ZN ^	NAND_MSF	0.005	0.012	0.424
ar1_ring_arb[31].ring_module1_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.014	0.025	0.437
ar1_ring_arb[31].ring_module1_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.015	0.040	0.452
ar1_ring_arb[31].ring_module1_g7	A2 ^ ->Z ^	AN2D1BWP30P140	0.022	0.062	0.474
external_hs[31].exhs_NDSR1	R ^ ->Q v	NAND_SR	0.015	0.078	0.490
external_hs[31].exhs_NRMSF2	I v ->ZN ^	NOR_MSF	0.027	0.104	0.516
encoder1_g1532	A1 ^ ->ZN ^	INR4D0BWP30P140	0.074	0.178	0.590
encoder1_g1503	A1 ^ ->ZN ^	IND2D1BWP30P140	0.035	0.214	0.626
encoder1_g1479	B3 ^ ->ZN v	INR4D0BWP30P140	0.020	0.233	0.646
encoder1_g1473	B2 v ->ZN ^	IND4D1BWP30P140	0.026	0.259	0.671
encoder1_g1468	A4 ^ ->Z ^	OR4D1BWP30P140	0.029	0.288	0.700
	req_out ^		0.000	0.288	0.700

Table 5.7: AER ring transmitter token loop delay Innovus timing report

Instance	Arc	Cell	Delay	Arrival Time
ar1_ring_arb[31].ring_module1_DFF1_Q_reg	CP ^			0.000
ar1_ring_arb[31].ring_module1_DFF1_Q_reg	CP ^ ->Q v	DFCND1BWP30P140	0.073	0.073
ar1_g15	I v ->ZN ^	INVD0P7BWP30P140	0.014	0.087
ar1_ring_arb[0].ring_module1_g12	A1 ^ ->Z ^	XOR2UD1BWP30P140	0.018	0.105
ar1_ring_arb[0].ring_module1_mutex1_NDSR1	R ^ ->Q v	NAND_SR	0.021	0.126
ar1_ring_arb[0].ring_module1_mutex1_NRMSF1	I v ->ZN ^	NOR_MSF	0.016	0.142
ar1_ring_arb[0].ring_module1_g8	A1 ^ ->ZN ^	INR2D0BWP30P140	0.022	0.164
ar1_ring_arb[0].ring_module1_DFF1_Q_reg	CP ^	DFCND1BWP30P140	0.000	0.164

Table 5.8: AER tree transmitter worst delay Innovus timing report

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	req[31] ^			0.000	0.335
external_hs[31].exhs_NRSR1	R ^ ->Q v	NOR_SR	0.006	0.006	0.342
external_hs[31].exhs_NDMSF1	I v ->ZN ^	NAND_MSF	0.004	0.010	0.346
at1_tree_lv1[15].xarb1_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.013	0.024	0.359
at1_tree_lv1[15].xarb1_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.015	0.039	0.374
at1_tree_lv1[15].xarb1_g35	A1 ^ ->Z ^	CKAN2D1BWP30P140	0.024	0.062	0.398
at1_tree_lv1[15].xarb1_g33	A2 ^ ->Z ^	OR2D1BWP30P140	0.020	0.082	0.418
at1_tree_lv2[7].xarb2_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.013	0.095	0.431
at1_tree_lv2[7].xarb2_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.015	0.110	0.445
at1_tree_lv2[7].xarb2_g35	A1 ^ ->Z ^	CKAN2D1BWP30P140	0.023	0.133	0.469
at1_tree_lv2[7].xarb2_g33	A2 ^ ->Z ^	OR2D1BWP30P140	0.023	0.156	0.492
at1_tree_lv3[3].xarb3_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.014	0.170	0.505
at1_tree_lv3[3].xarb3_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.015	0.185	0.520
at1_tree_lv3[3].xarb3_g35	A1 ^ ->Z ^	CKAN2D1BWP30P140	0.024	0.209	0.544
at1_tree_lv3[3].xarb3_g33	A2 ^ ->Z ^	OR2D1BWP30P140	0.020	0.229	0.564
at1_tree_lv4[1].xarb4_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.013	0.242	0.577
at1_tree_lv4[1].xarb4_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.015	0.257	0.592
at1_tree_lv4[1].xarb4_g35	A1 ^ ->Z ^	CKAN2D1BWP30P140	0.025	0.281	0.617
at1_tree_lv4[1].xarb4_g33	A1 ^ ->Z ^	OR2D1BWP30P140	0.020	0.302	0.637
at1_mutex1_NDSR1	S ^ ->QN v	NAND_SR	0.013	0.315	0.650
at1_mutex1_NRMSF2	I v ->ZN ^	NOR_MSF	0.026	0.341	0.677
at1_tree_lv4[1].xarb4_symCele1_g6	A1 ^ ->Z ^	OR2D1BWP30P140	0.021	0.363	0.698
at1_tree_lv4[1].xarb4_symCele1_NDSR1	S ^ ->Q v	NAND_SR	0.007	0.370	0.705
at1_tree_lv4[1].xarb4_symCele1_NRMSF1	I v ->ZN ^	NOR_MSF	0.015	0.385	0.720
at1_tree_lv4[1].xarb4_symCele1_g4	I ^ ->ZN v	INVD1BWP30P140	0.018	0.402	0.738
at1_tree_lv3[3].xarb3_symCele1_g6	A1 v ->Z v	OR2D1BWP30P140	0.022	0.424	0.760
at1_tree_lv3[3].xarb3_symCele1_NDSR1	S v ->Q v	NAND_SR	0.015	0.439	0.774
at1_tree_lv3[3].xarb3_symCele1_NRMSF1	I v ->ZN ^	NOR_MSF	0.015	0.454	0.789
at1_tree_lv3[3].xarb3_symCele1_g4	I ^ ->ZN v	INVD1BWP30P140	0.022	0.476	0.811
at1_tree_lv2[7].xarb2_symCele1_g6	A1 v ->Z v	OR2D1BWP30P140	0.025	0.500	0.835
at1_tree_lv2[7].xarb2_symCele1_NDSR1	S v ->Q v	NAND_SR	0.015	0.515	0.850
at1_tree_lv2[7].xarb2_symCele1_NRMSF1	I v ->ZN ^	NOR_MSF	0.015	0.530	0.866
at1_tree_lv2[7].xarb2_symCele1_g4	I ^ ->ZN v	INVD1BWP30P140	0.021	0.551	0.887
at1_tree_lv1[15].xarb1_symCele1_g6	A1 v ->Z v	OR2D1BWP30P140	0.023	0.575	0.910
at1_tree_lv1[15].xarb1_symCele1_NDSR1	S v ->Q ^	NAND_SR	0.006	0.581	0.917
at1_tree_lv1[15].xarb1_symCele1_NRMSF1	I ^ ->ZN v	NOR_MSF	0.004	0.585	0.920
at1_tree_lv1[15].xarb1_symCele1_g4	I v ->ZN ^	INVD1BWP30P140	0.016	0.601	0.936
external_hs[31].exhs_NDSR1	R ^ ->Q v	NAND_SR	0.020	0.621	0.956
external_hs[31].exhs_NRMSF2	I v ->ZN ^	NOR_MSF	0.026	0.647	0.982
g3347	A2 ^ ->Z ^	OR2D1BWP30P140	0.028	0.675	1.010
g3298	A2 ^ ->ZN v	ND4D1BWP30P140	0.025	0.699	1.035
g3287	B v ->ZN ^	OAI31D1BWP30P140	0.020	0.720	1.055
g3283	A3 ^ ->ZN v	AOI32D1BWP30P140	0.024	0.743	1.079
g3277	B3 v ->ZN ^	INR4D0BWP30P140	0.040	0.783	1.118
g3274	B ^ ->ZN v	AOI31D1BWP30P140	0.024	0.807	1.142
g3272	B3 v ->ZN ^	OAI33D1BWP30P140	0.057	0.864	1.199
	req_out ^		0.001	0.865	1.200

5.4 Verification Flow

The goal of the verification is to generate the SPICE file of the designed AER transmitter, which includes not only the transistors but also the RC extraction of the wires. With the SPICE, the functionality of the transmitter can be verified and the metastability conditions of the SR-latches can be checked in the Spectre simulator. The inputs of the generation flow are a GDS file and a Verilog file, one describes its geometric information and the other describes its logical expression. The two files are all generated by the Innovus at the end of the proposed design flow.

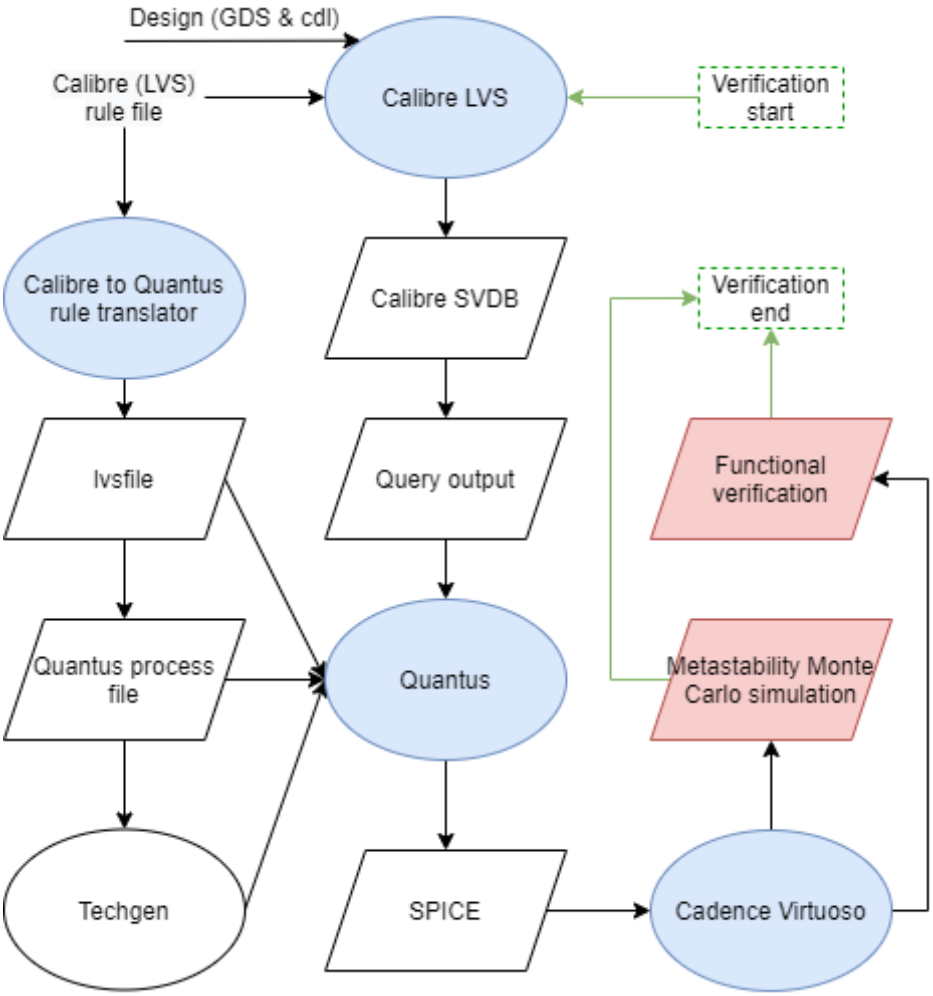


Figure 5.5: Verification flow chart

There are five tools/commands involved in the GDS to SPICE conversion, they are:

- **v2lvs**: a Calibre command to convert the Verilog Hardware Description Language to cdl, a Circuit Description Language. This cdl file is used to do Layout vs. Schematic (LVS) check with the GDS file.
- **LVS**: SPICE output from Quantus only applies to LVS input, here we use Calibre

LVS to generate a Calibre SVDB. It is important to make sure that the LVS check is clean so that a correct database can be generated.

- **Calibre Database Query**: The SVDB database is further processed by Calibre query to be accepted by Quantus.
- **Techgen**: For techfile generation, a lvsfile in the Quantus tech directory is mandatory in the Calibre flow. Our LVS rule deck is in an encrypted format, which can only be translated to Quantus rule by using Techgen compilation instead of a simple command in Calibre called "*calibre2qrc*".
- **Quantus QRC**: Once all needed files are available, the SPICE netlist of the AER transmitter is ready to be generated by the Quantus QRC.

The generated SPICE file of the transmitter contains resistors and capacitors with best, worst, and typical corners[2]. And it will be sent to the Spectre simulator to verify the functionality of the transmitter, as well as check the metastability of the SR-latches inside. Figure 5.6 shows the simulation environment of the established functional check verification flow. The "EACK" and the logic gates in the figure together represent the correct operation of the external hardware around the transmitter. In the simulation, whether the acknowledgement is corresponding to the request will be checked. Also, whether the address is corresponding to the grant signal, and whether the "req_out" is coming after the available address will be examined. The verification result shows that the design is correct in its function.

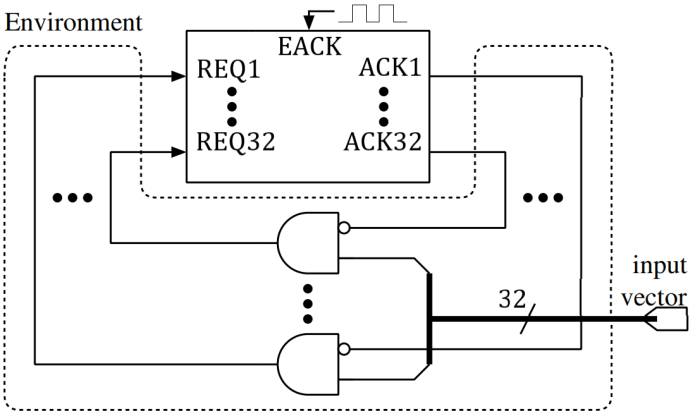


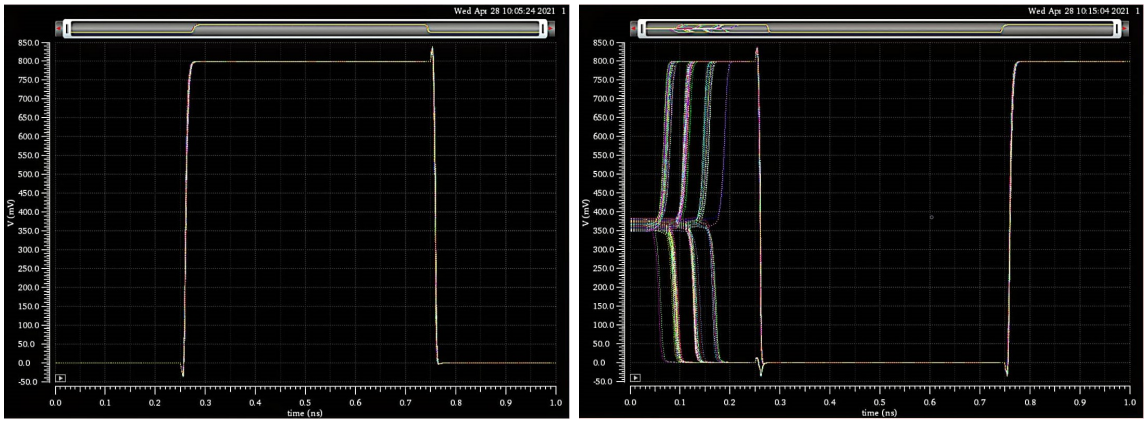
Figure 5.6: Simulation environment

[2]The three corners are controlled by three parameters. For instance, a resistor's value might be assigned as: "$R1 * 0.825893 + R2 * 0.825990 + R3 * 0.825995$". And the resistor is in the best corner when making $R1$ equals to 1 and $R2$ and $R3$ equal to 0.

5.5 Monte Carlo Simulation for Metastability

As mentioned, SR-latches are chosen to be the standard cells to construct asynchronous components such as C-elements and MUTEX in the design. And they are possible to enter a metastable state where there will be an uncertain voltage level produced. This will cause trouble to the circuit following it since the logic is also uncertain. Please note that the correct operation sequence of the transmitter is not able to drive the circuit into metastability logically. But because of gate delays, the precondition of metastability is possible to be formed and propagate to the SR-latches inside the circuit. To prevent that, metastability filters are necessary. To shorten the simulation time, a 4-bit transmitter is established to see the metastability and check the filters' performance in the Monte Carlo simulation.

Take NOR SR-latch as an example, the metastability for NOR SR-latch occurs when there is no previous state for 00 input. For example, if at time zero, the 00 input is given to the circuit, this will drive the cell into metastability. The consequence of the metastability can be observed through Monte Carlo simulation as Figure 5.7. When the NOR SR-latch enters the metastability as Figure 5.7b, its uncertain voltage level will be stabilized to either logic 1 or logic 0. And it is unpredictable in the 100 times simulation.

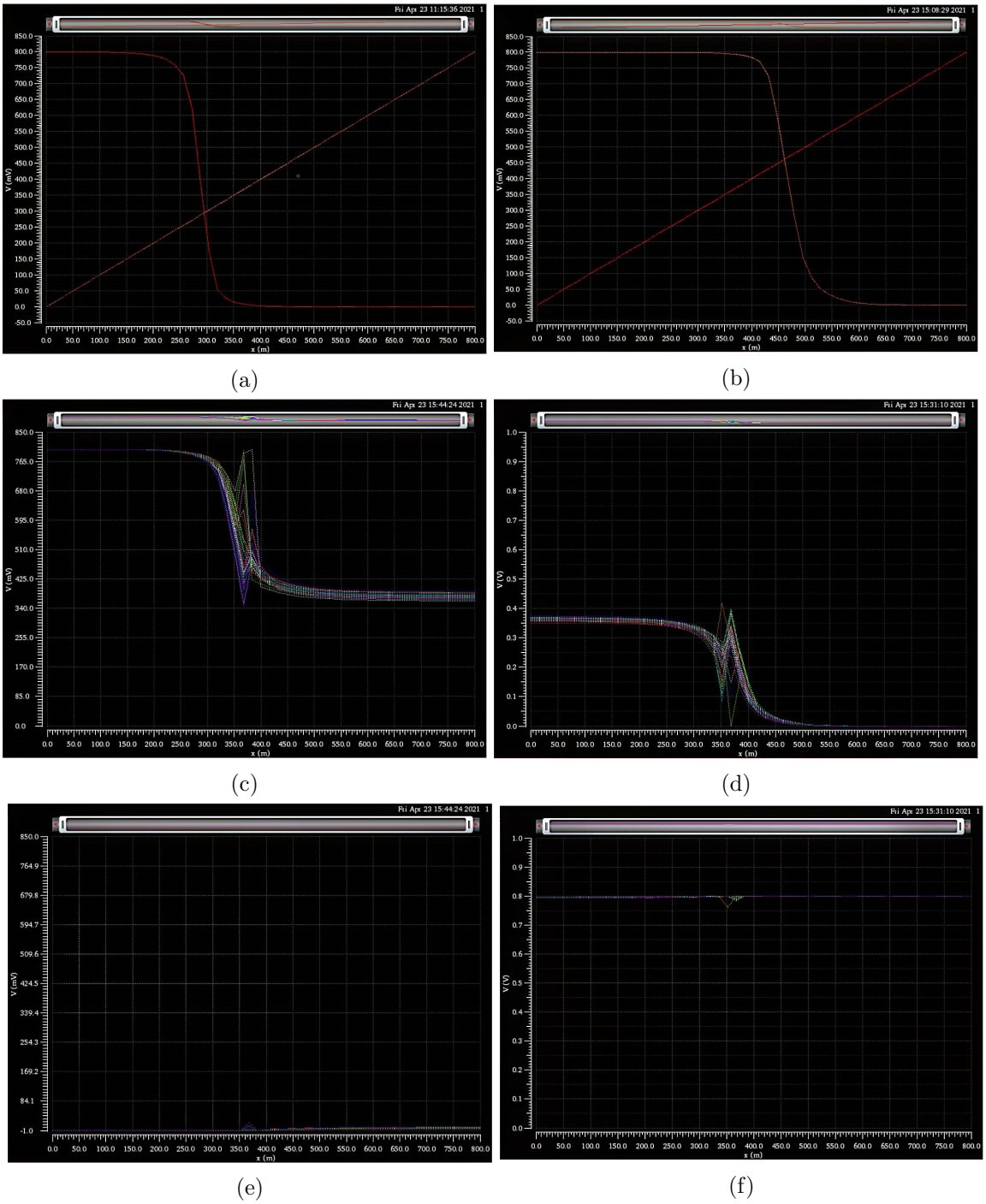


(a) (b)

Figure 5.7: Monte Carlo transit simulation of NOR SR-latch 100 times: (a) with previous state, (b) without previous state

The metastability filters can be built based on transistors or logic gates. Here it is based on logic gates of 4-input NAND and NOR. The filters are created by simply connecting the input pins of the 4-input NAND and NOR gates. By doing so, they act like a skewed inverter. The NAND and NOR gates are from the TSMC 28nm CMOS technology library. For the NAND SR-latch, we use NOR MSF as a filter. And for the NOR SR-latch, we use NAND MSF. Figure 5.8 explains the reason. Take NAND SR-latch as an example, the MSF for it, which is the NOR MSF, has DC transfer characteristics as Figure 5.8a. The switching threshold of the gate is lower than the middle voltage, which is $0.4V$. NAND SR-latch's DC characteristics (Figure 5.8c) show the "Q" pin of the latch falls to around $0.35V$ instead of 0 when getting

into metastability. For this range of latch output, the above presented NOR MSF is suitable to filter the metastable voltage all to 0 as Figure 5.8e. This is similar to NOR SR-latch and NAND MSF as figures on the right side of Figure 5.8.



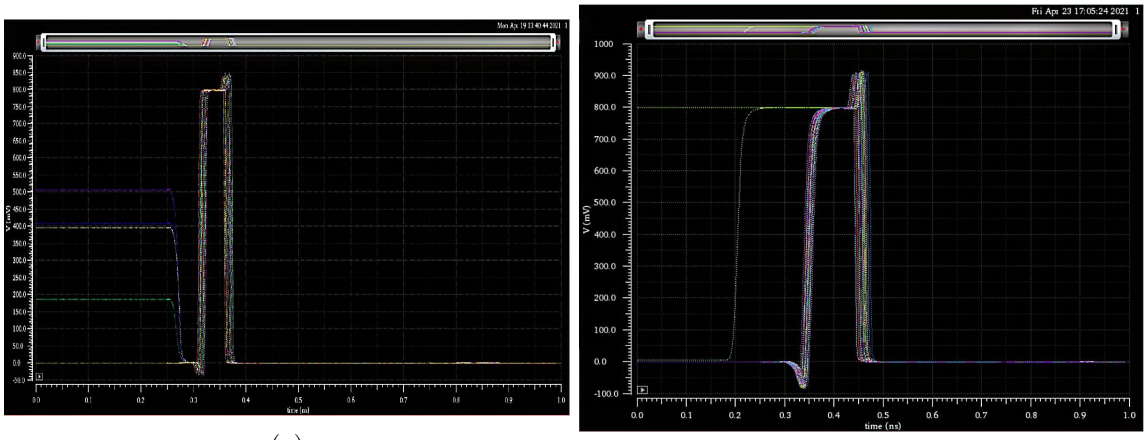
(a) (b)

(c) (d)

(e) (f)

Figure 5.8: MSF DC transfer characteristics: (a) NOR MSF, (b) NAND MSF; SR-latch DC metastability: (c) NAND SR-latch, (d) NOR SR-latch; SR-latch DC metastability after filtering: (e) NAND SR-latch, (f) NOR SR-latch

The designed MSFs are installed on the locations where there is a latch. It effectively prevents the occurrence of metastability. Two Monte Carlo simulations are done through GUI to plot the results visually as Figure 5.9. A precondition of making the latch inside the external handshake module metastable is produced by the simulator. Out of 30 times simulations, 4 times metastable states are observed. And they are restored to logic 0 or 1 after inserting the filters.



(a) (b)

Figure 5.9: Metastability of the 4-input transmitter: (a) without MSF, (b) with MSF

In a Monte Carlo test, more times of simulations guarantee a more reliable design. As the metastability can affect the output of the transmitter and lead the voltage level to uncertainty, more simulation data shall be generated to evaluate that whether the design can overcome the metastability under different PVT conditions. To fasten the speed of the test, the GUI method is abandoned and the command line method is used to execute the simulation. Another three Monte Carlo simulations are done to the three extreme cases: tt0p8v25c, ff1p05vm40c, and ss0p72vm40c, which corresponds to the typical, fast, and slow cases. For each of them, $10,000$ times of Monte Carlo simulations are done as Table 5.9. That amount of tests will generate huge temporary results data in the server that might cause disk quota exceeded problem. A script used to periodically remove those unnecessary data is created. The results show that there is no uncertain voltage level at all when initializing the latch with no previous state.

Table 5.9: Monte Carlo simulations of the 4-input transmitter under three PVT conditions

PVT	**Logic 0**	**Logic 1**	**Total Num**
tt0p8v25c	9,911	89	10,000
ff1p05vm40c	9,858	142	10,000
ss0p72vm40c	9,135	865	10,000

5.6 Transmitter Characterization Results

Before presenting the transmitter performance results, the tree arbiter module implemented with SR-latches in our design strategy is compared with several state-of-the-art arbiter designs as Table 5.10. It can be seen that the number of transistors of this work is larger than most of the proposed designs even not counting the MSF's transistors. But measured from the activation of the request signal to the propagation of the grant signal, our arbiter has the best delay under TSMC 28nm CMOS technology, it has an **11.54%** improvement than the arbiter who used to be the best in the IREQ1↑-IGR1↑ response time. The AER transmitters with different arbiter architectures are also reproduced and characterized using the developed flow. Here shows part of the characterization results as Table 5.11 and 5.12 to compare the three architectures under the typical corner, $0.8V$ VDD, and $25°C$.

Table 5.10: Arbiter feature comparison

		This work	[5]	[24]	[25]	[26]	[28]	[29]
Technology		28nm	65nm	65nm	65nm	65nm	65nm	65nm
Num of transistors		104	62	74	44	116	82	70
Four-way handshake		Yes	Yes	Yes	Yes	No	Yes	No
Use MUTEX		Yes	Yes	Yes	Yes	Yes	Yes	No
Gate MSF		Yes	Yes	Yes	No	Yes	Yes	-
Average response time	**IREQ1↑-OREQ↑**	59.5ps	53.68ps	206.18ps	-	50.81	74.80ps	143.05ps
		-	-10.84%	71.14%	-	-17.10%	20.45%	58.41%
	IREQ1↑-IGR1↑	101.5ps	187.62ps	249.48ps	-	295.57	202.75ps	114.74ps
		-	45.90%	59.32%	-	65.66%	49.94%	11.54%

Table 5.11: Performances of AER transmitters with tree arbiter module I and II

Inst Name	**Inst Count**	**Total Area**	**Inst Name**	**Inst Count**	**Total Area**
Tree: This work	768	1264.914	Tree: [5]	467	826.308
Total Power	**(mW)**		**Total Power**	**(mW)**	
Total Internal Power	0.022	67.172%	Total Internal Power	0.017	72.982%
Total Switching Power	0.007	19.457%	Total Switching Power	0.004	17.322%
Total Leakage Power	0.004	13.371%	Total Leakage Power	0.002	9.696%
Total Power	0.033		Total Power	0.023	
Worst slack	0.335		**Worst slack**	0.299	
Beginpoint	**Endpoint**	**Arrival Time (ns)**	**Beginpoint**	**Endpoint**	**Arrival Time (ns)**
req[31] (∧) triggered by leading edge of '@'	req_out (∧)	0.865	req[31] (∧) triggered by leading edge of '@'	req_out (∧)	0.901
req[3] (∧) triggered by leading edge of '@'	req_out (∧)	0.859	req[28] (∧) triggered by leading edge of '@'	req_out (∧)	0.900
req[10] (∧) triggered by leading edge of '@'	req_out (∧)	0.853	req[17] (∧) triggered by leading edge of '@'	req_out (∧)	0.899
req[14] (∧) triggered by leading edge of '@'	req_out (∧)	0.851	req[9] (∧) triggered by leading edge of '@'	req_out (∧)	0.899
req[28] (∧) triggered by leading edge of '@'	req_out (∧)	0.851	req[13] (∧) triggered by leading edge of '@'	req_out (∧)	0.892

The tables list the area, power distribution, worst slack, and the five worst delay propagation paths. The result shows that the arbiter module implemented using SR-latches has slightly better timing performance than Subbarao's. The worst path delay

of the former is 0.036ns faster than that of the latter. But Subbarao's arbiter has a smaller area and lower power consumption. This is because its architecture allows merging the C-elements into the logic and loops, so that eliminates the necessity of using MSF, which costs area and power. But the propagation delay of the MSF is shorter than a logic gate, thus making the SR-latches implemented architecture a bit faster in speed. Because the delay is the main parameter that this design cares about, it can be concluded that the former architecture is better for our specification.

Table 5.12: Performance of AER transmitter with lazy token ring arbiter module

Inst Name	**Inst Count**	**Total Area**
Ring	476	912.618
Total Power	**(mW)**	
Total Internal Power	0.016	70.730%
Total Switching Power	0.004	17.389%
Total Leakage Power	0.003	11.881%
Total Power	0.023	
Worst slack	0.412	
Beginpoint	**Endpoint**	**Arrival Time (ns)**
req[31] (∧) triggered by leading edge of '@'	req_out (∧)	0.288
req[31] (∧) triggered by leading edge of '@'	ad[0] (∧)	0.283
req[22] (∧) triggered by leading edge of '@'	req_out (∧)	0.279
req[23] (∧) triggered by leading edge of '@'	req_out (∧)	0.274
req[19] (∧) triggered by leading edge of '@'	req_out (∧)	0.270
Token loop delay	5.132	

Not the same as the reported conclusion in Section 2.2.4, the power consumption of the AER transmitter with tree arbiter is larger than that of with ring arbiter. The area of the tree transmitter varies with the tree arbiter module architecture, so it is not comparable. But at least the chosen tree transmitter's area is larger than that of the ring. The delay of the tree is larger than that of the ring, but the reported ring delay paths are the best-case delay paths. They are measured by tracking the signal from the request to the acknowledgement pin without considering the token loop delay. The token loop delay for the ring is very large. If we assume the loop delay can be averaged, (5.1) can be used to analyse which is better in delay performance. Where in the equation, D_{token_loop} is the token loop delay, N_{bits} is the number of bits, D_{ring_worst} is the worst best-case delay of the ring, and D_{tree_worst} is the worst delay of the tree. x belongs to the set of natural numbers.

$$\frac{D_{token\_loop}}{N_{bits}} \times x + D_{ring\_worst} \leq D_{tree\_worst} \quad (x \in \mathbb{N}) \tag{5.1}$$

By solving the equation, $x \leq 3.598 \approx 3$. This means the delay of when the request signal coming within the next three ring modules of the module that the token is currently located will be smaller than that of the tree. If assuming the other 31 request signals have an equal probability to occur next, then the probability of the next ring

delay is smaller than the delay when arbitrating using the tree is 3 over 31; and that of it has a larger delay is 28 over 31. According to this, it can be concluded that the timing performance of the ring is not better than that of the tree. The delay of the ring arbiter is uncertain considering different locations of the token and request occurrence probability. A certain and averaged delay is preferred for the project. Thus, the tree is of more value to be finally implemented.

The characterization results of the SR-latch tree AER transmitter under different PVTs are generated as Table 5.13. Firstly, the LIB files of asynchronous cells are generated by Liberate at different PVTs. After that, Genus synthesizes the AER transmitter under the PVT condition of tt0p8v25c[3]. The synthesized circuit will be taken as input to the Innovus to generate a report under different PVTs. And because we have the constraint, especially in the SS corner, the circuits will be synthesized to have a larger area to fulfil the requirement of delay. Here only presents the table of three parameters: total area, total power, and worst delay. Other information is also generated and stored. To conclude, Innovus reports that the delay falls within a reasonable range of from 0.475 to 1.150ns.

Table 5.13: Characterization result under different PVTs

	Total Area	**Total Power (mW)**	**Worst Delay (ns)**		**Total Area**	**Total Power (mW)**	**Worst Delay (ns)**
ff0p88vm40c	1264.788	0.037	0.618	ss0p81v0c	1264.914	0.029	1.031
ff0p88v0c		0.047	0.615	ss0p81v125c		0.090	0.936
ff0p88v125c		0.634	0.605	ss0p9vm40c		0.036	0.806
ff0p99vm40c		0.049	0.510	ss0p9v0c		0.037	0.796
ff0p99v0c		0.065	0.516	ss0p9v125c		0.115	0.781
ff0p99v125c		0.888	0.525	tt0p8v25c	1264.914	0.033	0.865
ff1p05vm40c		0.058	0.475	tt0p8v85c		0.078	0.834
ff1p05v0c		0.078	0.479	tt0p9v25c		0.044	0.683
ss0p72vm40c	1547.532	0.032	1.150	tt0p9v85c	1264.788	0.107	0.682
ss0p72v0c	1380.960	0.029	1.122	tt1v25c		0.057	0.578
ss0p72v125c	1275.624	0.071	1.108	tt1v85c		0.143	0.585
ss0p81vm40c	1265.418	0.029	1.064				

[3]This is the PVT expression of TSMC library, for example, ff1p05vm40c means FF corner, 1.05V VDD, and $-40°C$.

6 Conclusion

6.1 Conclusion

In this thesis, a novel asynchronous library establishment strategy of using SR-latches as standard cells to build AER communication circuits has been proposed. The library correctness, strategy feasibility, as well as design performance, are simulated and evaluated through a proposed asynchronous back-end design flow and a SPICE verification flow. The two flows are developed based on the existing digital design flow but with some adaptations to fit asynchronous design.

Just like computer arithmetic uses the add and shift operations to perform minus, multiply, divide, and more complex operations; like digital circuit uses AND, OR, inverter, buffer to construct more complex combinational systems; the most basic components in building asynchronous circuits are our targets to be found and researched so that with these components and other logic gates, functions of asynchronous circuits can be achieved without laborious work. SR-latch, in our exploration, has been proven to be able to establish the AER transmitter circuit. Its asynchronous sequential logic can support the correct AER communication operation.

With the SR-latches implemented asynchronous modules, AER transmitters based on three different arbiter architectures are established and compared. The simulation results show that with our library design strategy, the tree arbiter module with the most original and traditional architecture can be compared favourably with several state-of-the-art arbiter architectures. This can be further validated by the transmitter performance comparisons, where two state-of-the-art arbiter architectures are reproduced and implemented under our technology.

6.2 Future Work

Future works of this thesis can be extended in the following three aspects:

1. The logic of SR-latches is enough to support constructing asynchronous circuits built by C-elements. However, the latches built by standard cells only have one driving strength provided in our library, which is selected with the lowest delay. The asynchronous library can be complemented by adding more driving strength choices. And the logic of SR-latch can be made recognizable to the synthesize tool so that the tool will automatically pick the most suitable one according to its input/output capacitance information, etc.
   Even though SR-latches can construct C-elements that support many asynchronous logic implementations. Other logic requires other specific component sets. If such asynchronous circuits can be structurally decomposed to find their

basic constructing elements. These elements can be customized and characterized as a library to support asynchronous design using digital design flow.

2. The AER transmitter design in our work only considers logic gate-level implementation. There is a paper that shows that transistor-level design is the most economical in terms of silicon area [43]. A very simple example is the metastability filter. The gate-level implementation utilizes the skewed inverting property of the NOR and NAND gates. Such property can be easily achieved by a proper sized NMOS and a PMOS at the transistor level. However, it is not as easy as simply sizing two transistors and connecting them as an inverter, since its switching voltage can be varied according to different PVT conditions, possibly making preventing metastability fail in that case. Hence, the next step of the work in this direction can be establishing a well-designed transistor-level asynchronous library and uses simulations to validate the functions and performances of the cells.

3. Because of circuit decomposition, manually creating scripts for characterizing asynchronous library is not laborious work for this thesis. However, it should be admitted that making this flow automated to be able to universally characterize asynchronous cells is better when characterizing a library with more complex logic, especially when the cell arcs are too complex to be described to the tool.
The proposed back-end design flow and SPICE verification flow are developed based on the existing digital flow but with several changes to adopt the asynchronous design. This flow is automated for our design, namely from generating asynchronous library under different PVTs, synthesizing, to generating the final output of the placed and routed design, the operation of the tool can be executed by one or two command lines. However, if the target design is changed from the current AER transmitter. The strategy of back-end design flow might also be changed with it. For instance, the combinational loops cannot be broken automatically by the tools since for a random and unknown (to the tool) design, its interest signal propagation paths are also unknown if the designer does not specify them. Nonetheless, asynchronous circuits with the same type/function/implementation might have common grounds in their architectures and interest paths. From this point, it might be possible to also make the back-end design flow automated.

Bibliography

[1] Francisco Gomez-Rodriguez, Rafael Paz, Alejandro Linares-Barranco, Manuel Rivas, L Miro, Saturnino Vicente, Gabriel Jimenez, and Antón Civit. Aer tools for communications and debugging. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE, 2006.

[2] Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-based neuromorphic systems*. John Wiley & Sons, 2014.

[3] V. B. Marakhovsky. Logic design of asynchronous circuits. https://elib.spbstu.ru/dl/1945.pdf/download/1945.pdf, Slides on the course. CS&SE Department, SPbPU. Accessed: 21-May-2021.

[4] Yu Liu, Xuguang Guan, Yang Yang, and Yintang Yang. An asynchronous low latency ordered arbiter for network on chips. In *2010 Sixth International Conference on Natural Computation*, volume 2, pages 962–966. IEEE, 2010.

[5] Girish A Subbarao and Philipp D Häfliger. Design and comparison of synthesizable fair asynchronous arbiter. In *2020 18th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 122–125. IEEE, 2020.

[6] Masashi Imai, Tomohiro Yoneda, and Takashi Nanya. N-way ring and square arbiters. In *2009 IEEE International Conference on Computer Design*, pages 125–130. IEEE, 2009.

[7] Xu-Guang Guan, Xing-Yuan Tong, and Yin-Tang Yang. Quasi delay-insensitive high speed two-phase protocol asynchronous wrapper for network on chips. *Journal of Computer Science and Technology*, 25(5):1092–1100, 2010.

[8] Nabil Imam and Rajit Manohar. Address-event communication using token-ring mutual exclusion. In *2011 17th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 99–108. IEEE, 2011.

[9] Hesham Mostafa, Federico Corradi, Marc Osswald, and Giacomo Indiveri. Automated synthesis of asynchronous event-based interfaces for neuromorphic systems. In *2013 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4. IEEE, 2013.

[10] Yanhai Jiang, Hui Wu, and Hong Chen. A novel characterization method of click element based on cutting feedback loops in standard cell library design. In *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, pages 1–2. IEEE, 2019.

[11] Matheus Trevisan Moreira, Carlos Henrique Menezes Oliveira, Ney Laert Vilar Calazans, and Luciano Copello Ost. Lichen: Automated electrical characterization of asynchronous standard cell libraries. In *2013 Euromicro Conference on Digital System Design*, pages 933–940. IEEE, 2013.

[12] Matheus Trevisan Moreira and Ney Laert Vilar Calazans. Design of standard-cell libraries for asynchronous circuits with the ascend flow. In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 217–218. IEEE, 2013.

[13] Alain J Martin and Mika Nystrom. Asynchronous techniques for system-on-chip design. *Proceedings of the IEEE*, 94(6):1089–1120, 2006.

[14] HC Brearley. Illiac ii-a short description and annotated bibliography. *IEEE Transactions on Electronic Computers*, (3):399–403, 1965.

[15] AJ Martin, A Lines, R Manohar, M Nystrm, P Penzes, R Southworth, U Cummings, and TK Lee. The design of an asynchronous mips r3000 processor. In *Proceedings of the 17th Conference on Advanced Research in VLSI*, 1997.

[16] Alain J Martin, Mika Nystrom, and Catherine G Wong. Three generations of asynchronous microprocessors. *IEEE Design & Test of Computers*, 20(6):9–17, 2003.

[17] Alain J Martin. Compiling communicating processes into delay-insensitive vlsi circuits. *Distributed computing*, 1(4):226–234, 1986.

[18] Massimo Antonio Sivilotti. *Wiring considerations in analog VLSI systems, with application to field-programmable networks*. PhD thesis, California Institute of Technology, 1991.

[19] Rodney J Douglas, Misha A Mahowald, and Kevan AC Martin. Hybrid analog-digital architectures for neuromorphic systems. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1848–1853. IEEE, 1994.

[20] Kwabena A Boahen. Point-to-point connectivity between neuromorphic chips using address events. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(5):416–434, 2000.

[21] Henrik Eriksson, Per Larsson-Edefors, Tomas Henriksson, and Christer Svensson. Full-custom vs. standard-cell design flow: an adder case study. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 507–510, 2003.

[22] Oleg Petlin and Steve Furber. *Designing C-elements for testability*. University of Manchester, Department of Computer Science, 1995.

[23] David John Kinniment and Alex Yaklovlev. *Synchronization and arbitration in digital systems*. Wiley Online Library, 2007.

[24] Jens Sparsø and S Furber. Principles of asynchronous circuit designa systems perspective, 2001. *Google Scholar Google Scholar Digital Library Digital Library*.

[25] KA Boahen. A burst-mode word-serial address-event link ieee trans. on circuits and systems, 2004.

[26] Syed Rameez Naqvi and Andreas Steininger. A tree arbiter cell for high speed resource sharing in asynchronous environments. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[27] Alberto Ghiribaldi, Davide Bertozzi, and Steven M Nowick. A transition-signaling bundled data noc switch architecture for cost-effective gals multicore systems. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 332–337. IEEE, 2013.

[28] Gabriele Miorandi, Davide Bertozzi, and Steven M Nowick. Increasing impartiality and robustness in high-performance n-way asynchronous arbiters. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 108–115. IEEE, 2015.

[29] Timothé Turko, Wilfried Uhring, Foudil Dadouche, and Laurent Fesquet. An asynchronous fixed priority arbiter for high througput time correlated single photon counting systems. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 765–768. IEEE, 2018.

[30] Self-timed and asynchronous design. https://ece.uwaterloo.ca/~mhanis/ece730/lecture6.pdf, Slides on the course. ECE730, UWaterloo. Accessed: 28-May-2021.

[31] Alain J Martin. Practical asynchronous circuits and tools. *IEEE Design & Test of Computers*, 19(4):108–108, 2002.

[32] Ken Stevens, Shai Rotem, Steven M Burns, Jordi Cortadella, Ran Ginosar, Michael Kishinevsky, and Marly Roncken. Cad directions for high performance asynchronous circuits. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 116–121, 1999.

[33] Ad Peeters and Kees Van Berkel. Synchronous handshake circuits. In *Proceedings Seventh International Symposium on Asynchronous Circuits and Systems. ASYNC 2001*, pages 86–95. IEEE, 2001.

[34] Daniel H. Linder and JH Harden. Phased logic: Supporting the synchronous design paradigm with delay-insensitive circuitry. *IEEE Transactions on Computers*, 45(9):1031–1044, 1996.

[35] Robert B Reese, Mitchell A Thornton, and Cherrice Traver. A coarse-grain phased logic cpu. In *Ninth International Symposium on Asynchronous Circuits and Systems, 2003. Proceedings.*, pages 2–13. IEEE, 2003.

[36] Jordi Cortadella, Alex Kondratyev, Luciano Lavagno, and Christos P Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):1904–1921, 2006.

[37] Nikolaos Andrikos, Luciano Lavagno, Davide Pandini, and Christos P Sotiriou. A fully-automated desynchronization flow for synchronous circuits. In *2007 44th ACM/IEEE Design Automation Conference*, pages 982–985. IEEE, 2007.

[38] Karl M Fant and Scott A Brandt. Null convention logic/sup tm: A complete and consistent logic for asynchronous digital circuit synthesis. In *Proceedings of International Conference on Application Specific Systems, Architectures and Processors: ASAP'96*, pages 261–273. IEEE, 1996.

[39] Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, and Alex Kondratyev. Asynchronous design using commercial hdl synthesis tools. In *Proceedings Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)(Cat. No. PR00586)*, pages 114–125. IEEE, 2000.

[40] Matheus T Moreira and Ney LV Calazans. Electrical characterization of a c-element with lichen. In *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pages 583–585. IEEE, 2012.

[41] Semiconductor Engineering. Power consumption: Components of power consumption. https://semiengineering.com/knowledge_centers/low-power/low-power-design/power-consumption/, 2020. Accessed: 13-June-2021.

[42] Cadence Learning & Support: Genus. How does genus handle combinational feedback loops, and how to identify, report and break the combinational feedback loops. https://support.cadence.com/apex/ArticleAttachmentPortal?id=a1Od0000000nWloEAE, 2020. Accessed: 13-June-2021.

[43] JP Murphy. Design of latch-based c-element. *Electronics letters*, 48(19):1190–1191, 2012.