Dean&Francis

ISSN 2959-6157

Research Progress of Asynchronous FIFO Design

Zhuocheng Wang

School of Physics, Sichuan University, Chengdu, China
wangzc@stu.scu.edu.cn

Abstract:

With the high-speed development of integrated circuits, digital systems are growing in size and complexity. In complex digital systems, different modules may run under different clock domains. For example, in a communication system, the data receiving module and the data processing module may be driven by different clocks, where it requires an effective data caching mechanism to coordinate data transfer between different clock domains. Asynchronous FIFO meets this need. It can securely store and transfer data between different clock domains, reducing the possibility of data loss and errors. This paper gives a comprehensive review of the design of asynchronous FIFO, and summarizes the characteristics and advantages of asynchronous FIFO in a metastable state, empty-full signal creation, low latency, high throughput, parameterizable design, and the performance of different design methods and application scenarios by analyzing several related research results. The purpose is to provide a reference for further research and application of asynchronous FIFO.

Keywords: Asynchronous FIFO; metastable state; empty-full signal; low latency; parameterizable design

1. Introduction

GALS (Globally Asynchronous Locally Synchronous) is an integrated circuit design method, and asynchronous FIFO (First In First Out) is an archetype. In a GALS system, different modules can run under separate local asynchronous clocks. This can avoid some problems caused by global clock synchronization, such as clock skew and clock distribution issues. Meanwhile, on a local scale, synchronous design can be used within each module to simplify design and improve performance.

As an important data cache structure, the design of asynchronous FIFO poses a higher challenge. In modern integrated circuits, data is often transmitted in different time domains, and asynchronous FIFO is mainly used for data transmission between different clock domains, which can effectively solve the problem of clock synchronization and ensure the correct storage and reading of data [1]. In this regard, asynchronous FIFO is more widely used than synchronous FIFO, such as in network communication, image processing, digital signal processing.

Presently, many research schemes have been explored and optimized for asynchronous FIFO, such as Gray code, parallel design, combination with SOC and NOC, parameterizable design, and interaction

with communication protocol including UART. These schemes improve the performance of asynchronous FIFO to varying degrees, but there are some shortcomings, which need further research and improvement.

This study hopes to explore the potential of their combination by analyzing the characteristics, advantages and defects of each study. This research deeply studies the existing schemes and the complementarity between them. It is expected to propose a more optimized asynchronous FIFO design scheme and provide a reference direction for the development of integrated circuit technology.

2. Background of Asynchronous FIFO

Design

The structure of asynchronous FIFO can be seen in fig. 1, which mainly consists of storage unit, read pointer, write pointer, full and empty signal. Data is written to the storage unit in FIFO order. The write pointer points to the current write position and the read pointer points to the current read position. When a write operation occurs, the write pointer increments and data is written to the storage unit. When a read operation occurs, the read pointer is incremented to read data from the storage unit [2]. By comparing the values of the write and read pointers, whether the FIFO state is full, empty, or in the middle state can be determined.

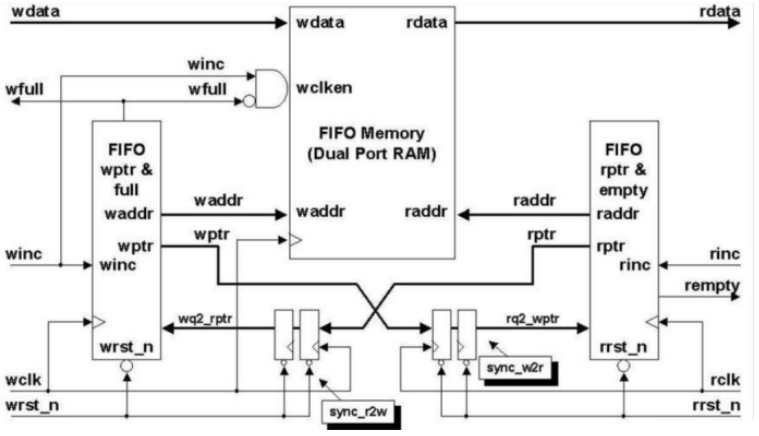

Fig. 1 Diagram of asynchronous FIFO[3]

At the beginning of asynchronous FIFO design, the problem of metastability is faced. Metastable is when the sampled signal jumps along the clock and the register is sampled to an intermediate value of the logical 0 and logical 1 reference voltages. Metastability gradually returns to a logical 0 or 1 over time, where it is impossible to predict whether it will become 0 or 1.

In general, the synchronous clock will not be metastable in the case of guaranteed setup and hold, which is also the reason why the synchronous clock does not need to turn Gray code. The phase relationship of asynchronous clocks can not be determined, so it is possible that the signal before synchronization just jumps along the counterpart clock. There is a probability of metastable state.

Asynchronous FIFO is an effective method to solve this issue. By using specific design structure and protocol, it can realize the secure transmission and storage of data between different clock domains. Asynchronous FIFO usually consists of the storage unit, read and write pointer management, and empty-full state judgment. Among them, the management of read and write pointers and the judgment of empty-full state are the key links of asynchronous FIFO design [3].

3. The Design of Asynchronous FIFO

3.1 Using Gray Code to Reduce Metastability

For the creation of empty and full signals, the address pointer is very crucial because the asynchronous FIFO has the problem of judging read empty or write full across the clock domain. The write and read pointer changes in sequence to ensure first-in-first-out [4]. When the Gray code is used, the FIFO status is determined by comparing read and write addresses, as shown in Table 1. If the highest and second highest bits of the address are the same, FIFO is empty. Likewise, if they are both different, FIFO is full.

ISSN 2959-6157

TABLE 1. The principle of state comparison

Gray Code	Write Address	Read Address	State
Empty FIFO	0 0000	0 0000	empty
Full	1 1000	0 0000	full
Empty	1 1000	1 1000	empty
Full	0 0000	1 1000	full
Empty	0 0000	0 0000	empty

The Gray code is used in the asynchronous clock address comparison because the orderly transformation of the Gray code changes only 1bit at one time. While in binary code, multiple bits can change 5 bits at the same time, such as 01111→10000. Therefore, the probability of metastability is reduced and the reliability of the signal is enhanced due to the Gray code. At the same time, cross-clock domains have read-slow-write-fast or read-fast-write-slow situations, the solution is 2 registers. Although it is time-consuming and the conservative judgment of empty-full states affects performance, it does not affect the function. In a word, generating empty-full status through Gray code pointer and using 2 registers can effectively reduce the probability of metastability and improve system stability [5].

However, the Gray code may also be metastable during the jump. Because the metastable code will eventually return to logical 0 or 1, the Gray code after metastability may also have two situations compared with the state before the jump. That is normal jump or no jump of the signal.

For the normal jump, it will not have any effect on the result. For the no jump, it will make the synchronized pointer more conservative. It will not appear the situation that it continues to write when FIFO is full or it continues to read when it is empty. Although there is some impact on the performance, the function is more reliable, which is also an important reason for choosing to use Gray code across the clock.

3.2 Low Latency and High Throughput

Traditional data transmission is normally a serial structure, such as UART protocol, with input data going through all cells to reach the output port. Hence, the cost of high throughput is often high latency.

As shown in fig. 2, the most prominent feature of Y. Xiao and R. Zhou's design is the parallel loop structure, in which each FIFO unit consists of a controller and a storage unit, so that the unit can directly communicate with the input and output ports through a common bus, without passing all intermediate cells. The extra movement of data from the input port to the output port can be eliminated [6]. Therefore, this cyclic architecture theoretically provides a low latency, high throughput that is almost independent of the number of FIFO stages, reducing time and power consumption [6].

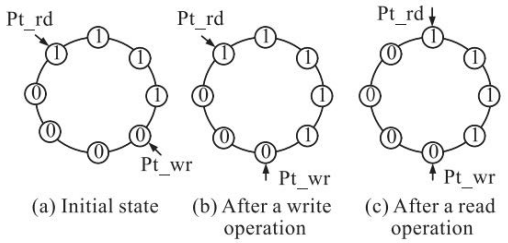

Fig. 2 Parallel circular asynchronous FIFO[6]

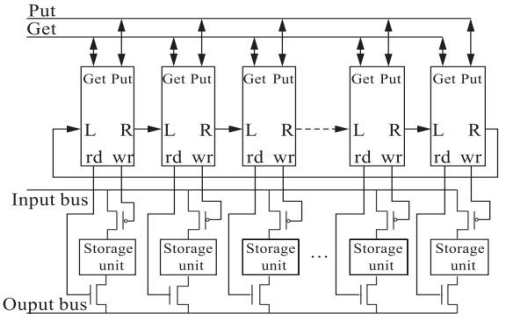

Fig. 3 An example of the protocol of FIFO[6]

For FIFO controllers, the study uses a monorail asynchronous protocol to simplify the design, which can be seen in fig. 3. Each unit controller only needs three C-gates, thus reducing the chip area.

Simulation by TSMC 0.25μm CMOS logic process proves that it does have very low latency, less than 581ps, and throughput above 2.2GHz, which can meet the needs of high-performance systems [6].

Through the parallel loop structure, the design can improve the throughput while ensuring that the delay does not improve with the increase of FIFO series. It demonstrates the stability in large-scale data storage and processing to a certain extent.

Nevertheless, this verification method relies on TSMC 0.25μmcomplementary metal-oxide-semiconductor logic process, which may have different performance under different processes.

3.3 FIFO Buffer Application on SOC

With the development of chips, more and more IP(Intellectual Property) needs to be integrated on the SOC(System on Chip). Asynchronous design does not involve complex clock distribution, so it can decrease the chip area of clock management on the SOC.

Because the asynchronous FIFO is designed with GALS and allows the integration of components with various time characteristics, it is friendly for SOC designs that need to integrate a variety of IPs involving different clock signals.

M. Menaka (2023) and Suruchi Chaturvedi (2024) designed the buffer architecture as a circular queue, and then compared the performance differences between synchronous and asynchronous designs in terms of power consumption, delay, throughput, and chip area [7]. Both were simulated using LTSpice on TSMC 180nm technology.

The power consumption comparison of synchronous and asynchronous circuit four-stage FIFO buffer design is shown in Table 2, concluding that the power consumption of asynchronous design is about 50% lower than that of synchronous design [8]. This can effectively show that in the application of FIFO buffer on SOC, the asynchronous design can greatly reduce power consumption.

Table 2. Efficiency metrics[8]

Performance Metric	Synchronous Design	Asychronous Design
Power dissipation	28.57μW	14.93μW
Cycle time	1ns	2ns
Throughput	1Gbits/s	0.5Gbits/s
Write Latency	395ps	194ps
Read Latency	387ps	1067ps
Transistor Count	804	733

This design performs well when it comes to write latency, but is flawed in terms of read latency, which is around 175% higher than that of synchronous design. The asynchronous design also lags behind the synchronous design in terms of throughput. Its characteristics may also be subject to process limitations of TSMC 180nm.

3.4 Asynchronous FIFO for PVT Changes

PVT refers to the changes in the Process, Voltage and Temperature of the integrated circuit manufacturing process. Traditional SOC has an increasing number of Processing Elements (PEs) on a single module, and they are in a state of communication with each other, sharing information, forming an information network, often called Network on Chip (NoC).

PEs communicate with each other by clock. As a result, for unbalanced clock distribution, data that violates the establishment/retention time of the clock edge may cause information loss or communication interruption due to changes in PVT. In addition, multiple clock domains typically consume plenty of power, and a single clock domain might span many square millimeters.

In order to solve these shortcomings, S. Abdel-Hafeez and M. Q. Quwaider proposed the design of energy-saving asynchronous FIFO memory based on handshake signal (request signal and acknowledgement signal) as shown in fig. 4. In this way, each PE has an asynchronous FIFO, while the FIFO has no clock signal. The next PE receives a request signal from the communication PE to work [9]. In response, it generates an acknowledgement signal to synchronize write/read operations within the FIFO [9].

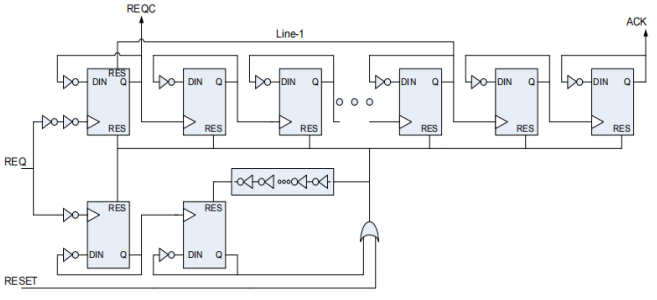

Fig. 4 Asynchronous signaling circuit[9]

ISSN 2959-6157

The design has a clear sequential structure, and the sequence of signals can be guaranteed by requesting confirmation signals. Compared with the effect of PVT on the clock, the its effect on the transistors after removing the clock is acceptable, and the metastability and delay caused by the clock signal are avoided.

The simulation results show that the data-access speed is fast. The asynchronous FIFO read operation can read data on the bus after 20-gate delays at the edge of the request signal, where each gate delay is about 0.02ns, so the read operation can read after 0.5ns at the edge of the request signal [9]. Similarly, write operations can push data at the memory array unit after a 12-gate delay, nearly 0.25ns, relative to the edge of the write request signal [9].

Table 3. Signal definitions[9]

Components	COMS Transistor Counts	Power Consumptions (mW)
Control Unit	72	0.04
Parallel Counter	460	0.27
Address Decoder	384	0.19
Empty-Full Flags	146	0.11
64-word x 64-bit SRAM 8T-Cell	33408	1.3

Table 3 shows that it can be considered as a potential choice technology for low-power applications, total power consumption at 65nm and 1V supply is only 2mW. In the case that the clock distribution will increase the power consumption, the design can greatly reduce the overall power consumption of the system. It can also avoid the influence of excessive current and heat on other performance of the system. At the same time, the design avoids the rigorous clock control to occupy the chip area.

However, new asynchronous circuits may increase the complexity of design and layout, and bring some difficulties to design and implementation.

3.5 Parameterizable Design and Modular Asynchronous FIFO

H. Ashour design realizes the parameterization of data interface width and memory depth. First of all, based on the design of the read and write pointer to form the empty-full signal in turn, it reflects the asynchronous FIFO design technology. Thus, it also demands that the read pointer and the write pointer can be parameterized, meaning the empty-full signal can also adapt to this point.

The design scheme can be parameterized through Verilog HDL code, and each module is configured separately and the code is independent. As a consequence, each module in the architecture has independent and self-contained functions, providing a modular interface for other system-level components, which can be reused in other systems [10]. The FIFO threshold can be reconfigured at run time. During FIFO instantiation, the HDL parameters, as shown in Table 4 for example, can be reallocated [10].

Table 4. Parameters[10]

Parameter	Description	Notes
DATA_SIZE	Determines the width of the wdata/rdate ports	
ADDR_SIZE	Determines the width of the write pointer, read pointer, near full threshold value and near empty threshold value	FIFO memory depth is calculated to be two to the power of value of this parameter

The parameterizable design can flexibly adjust the data interface width and memory depth according to different application scenarios, and the universality and adaptability of the design are greatly enhanced. At the same time, the function of modules in the architecture is independent, which is easy to maintain and expand, improving the reusability of the design and reducing the cost of development. In the meantime, the FIFO threshold can be reconfigured during running, so that it can meet the different requirements of different applications for empty-full signals. Fur-

thermore, it is suitable for rate matching, reconfigurable hardware and data flow application scenarios.

However, parameterized and modular designs may increase the complexity of the design. For instance, reconfiguring the threshold at run time may give rise to a certain performance cost, affecting the FIFO read and write speed. At the same time, modular design may occupy more hardware resources, especially on the FPGA platform with limited resources, which may affect the realization of other functional modules.

3.6 Buffering Characteristics of FIFO and Its Application in UART

UART (Universal Asynchronous Receiver/Transmitter) is a universal serial and asynchronous communication protocol. In UART, the sender converts parallel data into serial data, then adds start bit, data bit, check bit, stop bit, and other information. Then it is sent out bit by bit, as shown in fig. 5. The receiver detects the arrival of the data based on the start bit, receives the data bit by bit in an agreed format, and then converts the serial data into parallel data for subsequent processing.

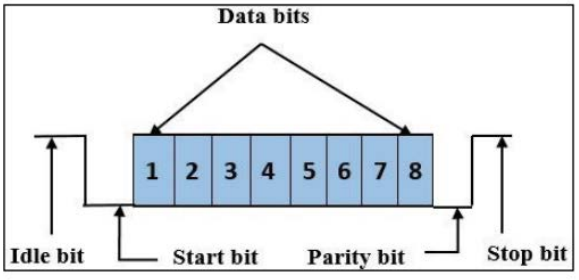

Fig. 5 Data frame format of UART Protocol[11]

Wenyu Wang (2023) uses FIFO buffer to solve the problem of UART data transmission, utilizing Verilog for design and asynchronous FIFO characteristics to comprehensively improve the performance of UART communication protocol. It also involves the implementation of FPGA platform.

The design uses the isolation characteristic of asynchronous FIFO to solve the problems of different clock domains, transmission rates and interface widths. Besides, it creates a data buffer to solve data loss or rewriting problems when devices are out of sync. In addition, this design establishes a communication network control system model to determine the optimal sampling times and upper limit to avoid large delays [11]. Moreover, the study adopts the optimal frequency doubling sampling technique, and the functional simulation is carried out by ModisticSE-64 [11]. This design can improve transmission efficiency, reduce bit error rate, and be more competitive in anti-interference.

Although the introduction of asynchronous FIFO and data buffer in UART improves the stability of data, the design of comprehensively improving the system performance may occupy certain hardware resources and increase the system cost. It does not explicitly mention the resource occupation situation, which may denote that it is not optimized in terms of resource utilization.

4. Conclusion

This paper deeply studies different aspects of asynchronous FIFO design, and finds that using Gray code pointer and two-stage synchronizer can reduce metastable probability and improve system stability, which is of great value in cross-clock domain design. The parallel cycle structure reduces time and power consumption and it has the characteristics of low delay and high throughput, meaning the delay does not increase with the increase of FIFO series, so as to meet the high performance requirements. Using asynchronous design on SOC can reduce the area of clock management chip and integrate different clock signal components. Simulation shows that the asynchronous design has low power consumption and low write delay, but it has read-delay defects and the results may be limited by 180nm process. The design based on handshake signal greatly reduces the impact of PVT, avoids metastability and delay, and has fast read and write speed, which is a potential choice for low-power applications. However, it may increase the difficulty of design and layout. Parametric and modular design can increase versatility and adaptability while reducing developmental costs, whereas it may increase the complexity, affecting the read-write speed, and occupying more hardware resources. The combination with UART improves the performance of the system as a whole. In the future, asynchronous FIFO designs may focus on reducing design complexity and power consumption while ensuring comprehensive performance, such as exploring new coding methods and structures. Meanwhile, it should improve the stability under different processes and combine with new communication protocols to further the versatility. As a consequence, asynchronous FIFO can adapt to more complex and changeable application requirements.

References

[1] Himanshu and C. Charan, A 16-Byte Asynchronous Gray Code FIFO Memory Using Verilog HDL for Real-Time Applications, 2024 2nd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT),2024: 707-711.

ISSN 2959-6157

[2] Z. Hao, L. Liu and B. Tian, The Principle and Applications of Asynchronous FIFO, 2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), 2023: 277-279.

[3] V. Patel, V. Mer, J. Patoliya and B. Soni, Design Implementation of Novel Asynchronous FIFO," 2023 IEEE International Symposium on Smart Electronic Systems (iSES), 2023: 292-295.

[4] Shivali and M. Khosla, Coverage of Meta-Stability Using Formal Verification in Asynchronous Gray Code FIFO, 2022 2nd International Conference on Intelligent Technologies (CONIT), 2022: 1-8.

[5] X. Ren, C. Li and K. Liu, Design and Implementation of High-Reliability FIFO Based on FPGA, 2023 IEEE 7th Information Technology and Mechatronics Engineering Conference (ITOEC), 2023: 480-484.

[6] Y. Xiao and R. Zhou, Low latency high throughout circular asynchronous FIFO, in Tsinghua Science and Technology, 2008, 13(6): 812-816.

[7] M. Menaka, A. K, T. R. Dinesh Kumar, S. K. G, S. W. M, and V. R. M, Asynchronous Circular Buffers based on FIFO for Network on Chips, 2023 International Conference on Circuit Power and Computing Technologies (ICCPCT), 2023: 1356-1361.

[8] S. Chaturvedi, S. M. N and R. Rao, Design of Asynchronous Circular FIFO Buffer for Asynchronous Network on Chips, 2022 International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2022: 66-71.

[9] S. Abdel-Hafeez and M. Q. Quwaider, A One-Cycle Asynchronous FIFO Queue Buffer Circuit, 2020 11th International Conference on Information and Communication Systems (ICICS), 2020: 388-393.

[10] H. Ashour, Design, simulation and realization of a parametrizable, configurable and modular asynchronous FIFO, 2015 Science and Information Conference (SAI), 2015: 1391-1395.

[11] W. Wang, Optimization of UART Communication Protocol Based on Frequency Multiplier Sampling Technology and Asynchronous FIFO, 2023 IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA),2023: 280-285.