Advances and Progress in Asynchronous FIFO Design

Qiao Lan *

School of Guangdong Polytechnic Normal University

* Corresponding Author Email: lq050119@outlook.com

Abstract. As integrated circuits continue to evolve, increasing the size and complexity of digital systems, in complex digital systems different modules may operate under different clock domains. For example, in a communication system, a data-receiving module and a data-processing module may be driven by separate clock sources. Therefore, an effective data caching mechanism is needed to coordinate data transfer between different clock domains. Asynchronous FIFOs fulfil this need by securely storing and transferring data between different clock domains, thereby reducing the risk of data loss and errors. By describing the design study of asynchronous FIFOs, this paper summarizes the features and advantages of asynchronous FIFOs in terms of sub-stability, creation of null-full signals, arbitrary adjustment of bit-width, and solving the problem of high latency through the analysis of several related research results, aiming to provide a reference for the further study and application of asynchronous FIFOs.

Keywords: Sub-stable, full and empty signals, low latency, parametric design, frame-level design.

1. Introduction

With the continuous progress of IC technology, the complexity and performance requirements of the system are increasing, and the design of asynchronous FIFO, as an important data caching structure, also poses a higher challenge to the design of asynchronous FIFO.

In modern integrated circuits, data is often transmitted in different time domains, and asynchronous FIFOs are mainly used for data transmission between different clock domains, which can effectively solve the clock synchronization problem and ensure the correct storage and reading of data [1]. Asynchronous FIFO is mainly composed of a memory cell, a read pointer, a write pointer, a full flag and an empty flag. Data is written to the memory cell in a first-in-first-out order, the write pointer points to the currently written position, and the read pointer points to the currently read position. When a write operation occurs, the write pointer is incremented and the data is written to the storage cell; when a read operation occurs, the read pointer is incremented, and the data is read from the storage cell. By comparing the values of the write pointer and the read pointer, you can determine whether the state of the FIFO is full, empty, or in an intermediate state.

By reading a lot of literature analysis, asynchronous FIFO design will face the problem of sub stability, which can be reduced by Gray code to reduce the sub stability, through the binary code is converted to Gray code, so that the asynchronous clock address will not appear more than one bit at the same time to change, which will help to solve the problem of sub stability, as well as the judgment of the empty and full state. Also encountered high latency problems, can be through the parallel loop structure, while improving throughput to ensure that the delay does not increase with the increase in the number of FIFO levels, can be a certain degree of stability in large-scale data storage and processing. The simplified design of the FIFO controller reduces chip area at the same time. To solve the problem of catching whole frame data by FIFO, an asynchronous FIFO based on Verilog HDL high-level programmable language is designed, which can be easily used for intermediate coaching of Internet data frames to ensure that Internet data frames are not easy to be lost under high throughput [1].

To achieve parameterizable design and modularity implemented through Verilog HDL code, each module is individually configured and code independent so that each module in the architecture has independent and self-contained functionality, providing modular interfaces to other system-level components that can be reused in other systems FIFO flag thresholds can be reconfigured at runtime, and these can be assigned when the FIFO is instantiated HDL parameters. The parameterized design

makes it possible to flexibly adjust the data interface width and memory depth according to different application scenarios, and the versatility and adaptability of the design are greatly increased; At the same time, the modules in the architecture are functionally independent, which is easy to maintain and extend, and can improve the reusability of the design and reduce the development cost. At the same time, the FIFO flag threshold can be reconfigured at runtime, making it possible to meet the different requirements of different applications for the empty-full flag, which is suitable for application scenarios such as rate matching, reconfigurable hardware and data flow.

Currently, there have been many research schemes explored and optimized for asynchronous FIFOs, such as the use of Gray code and parametric design. These schemes have improved the performance of asynchronous FIFOs to different degrees, but some shortcomings require further research and improvement.

2. Asynchronous FIFO Design Solution

2.1. Sub-stability reduction in asynchronous designs

2.1.1. Reducing Sub stability Using Gray Code

The judgment of empty and full state is the key to the operation of asynchronous FIFO. For the creation of empty and full signals, asynchronous FIFOs have the problem of judging read empty or write full across the clock domain, and the address pointer is critical. The address pointer is critical. The write pointer and read pointer change sequentially to ensure first-in-first-out. When using Gray Code, the read and write addresses are compared to determine the state of the FIFO, and the highest and second highest bits of the address are the same as the read empty, and if they are different, the write full.

When data is transferred from one clock domain to another if a clock that is asynchronous concerning the binary counter clock is used for sampling, it is necessary to consider which bit of the counter is changing in that range, and the worst-case scenario is that every bit may change, such as from FFFF to 0000, which is every bit may be in a suitable state.

In asynchronous clock address comparison using Gray code, because the order of the Gray code changes only 1 bit each time, there will not be more than one-bit change at the same time, for example, the binary code to Gray code conversion formula is as follows:

$$Gn = bn, gi = bi \oplus bi + 1 \; \forall i \neq n \tag{1}$$

The conversion formula for grey code to binary code is as follows:

$$Bn = gn, bi = bi + 1 \oplus gi \;\; \forall i \neq n \bigoplus_{i=0}^{n-1} \tag{2}$$

2.1.2. Asynchronous FIFO based on secondary synchronization chain and status bits

The secondary synchronization chain can greatly reduce the subs table phenomenon that occurs during the transmission of signals in different clock domains, and the output signal Q2 after the secondary synchronization chain can greatly reduce the appearance of sustainability. In the design of asynchronous FIFOs, the read address pointer is synchronized to the write clock domain by the second-stage synchronization chain, and then compared with the write address pointer to produce a full state; the write address pointer is synchronized to the read clock domain by the second-stage synchronization chain, and then compared with the read clock to produce an empty state. On the other hand, the write address pointer is synchronized by the secondary synchronization chain to the read clock domain and then compared with the read clock to generate a null state. If the address pointer has cycled the whole FIFO for an odd number of times, the marker position is 1, and if it has cycled for an even number of times, the marker position is 0. When the read and write address pointers are the same after the second-level synchronization chain and the two auxiliary marker bits are the same, it means that the read and write pointers have experienced the same number of cyclic addressing, and then the null state is valid; if the address pointers are the same and the status bits are opposite, it

means that the write pointer has cycled more than the read pointer, and then the full state is valid. If the address pointer is the same and the status bit is opposite, it means that the write pointer has cycled more than the read pointer, and the full state is valid [1].

The judgment is made in the following manner (fig 1):

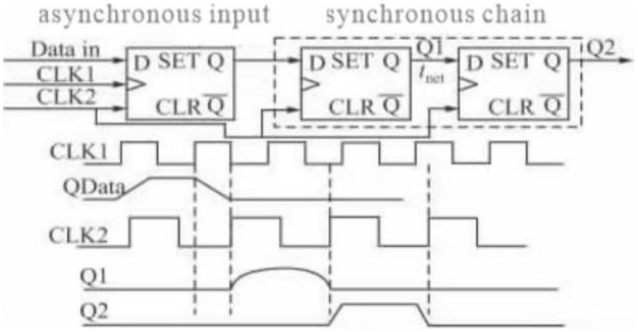

Fig 1. Secondary synchronous chain.

3. The solution to the high latency problem in asynchronous design

Much of traditional data transfer is serial in structure, such as the UART protocol, where input data passes through all cells to reach the output port, so the price of high throughput is often high latency.

As shown in Fig 2 and Fig 3. The most important feature of the Y. Xiao and R. Zhou design is the parallel loop architecture, where each FIFO cell consists of a controller and a memory cell, allowing the cell to communicate directly with the input and output ports over a common bus without going through all the intermediate cells, eliminating the process of moving data from the input port to the output port. As a result, this cyclic architecture provides a low latency, and high throughput almost independent of the number of FIFO stages. Reduced time and power consumption [2].

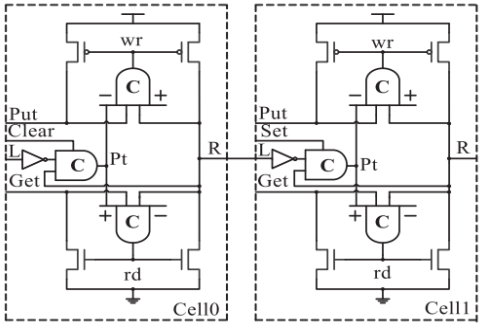

Fig 2. Parallel circular asynchronous FIFO.

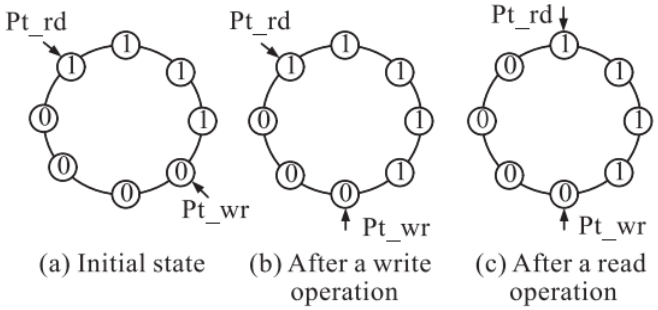

Fig 3. An example of the protocol of FIFO.

As shown in Fig 4, for FIFO controllers, the study simplifies the design using a single-rail asynchronous protocol that requires only three C gates per cell controller, thus reducing chip area.

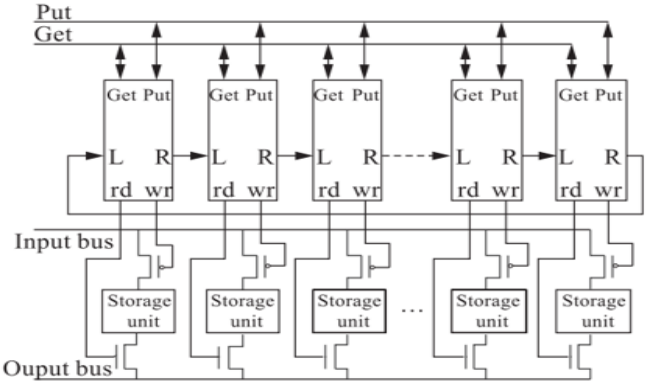

Fig 4. Exhibition of 3 C gates in FIFO controllers.

Simulation using TSMC's 0.25μm complementary metal-oxide semiconductor logic process demonstrates that it does have very low latency, less than 581ps, and throughput higher than 2.2GHz, which meets the needs of high-performance systems.

The design of the parallel loop structure, to improve throughput while ensuring that the delay does not increase with the increase in the number of FIFO levels, can be a certain degree of stability in large-scale data storage and processing. At the same time, the simplified design of the FIFO controller reduces chip area.

However, the validation methodology relies on TSMC's 0.25μm complementary metal-oxide semiconductor logic process, which may have different performance under different processes.

4. The solution to the caching problem in asynchronous design

As shown in Fig 5, to solve the problem of caching the whole frame data by FIFO, this asynchronous FIFO is implemented based on Verilog HDL high-level programmable language, which can be easily used for intermediate caching of Internet data frames to ensure that Internet data frames are not easy to lose in large throughput situations.

Frame-level FIFO is to store a packet as a minimum unit, including the following cases: (1) the packet is bad, then it is deleted from the FIFO by controlling the write RAM address and refreshing the end position of the previous packet; (2) the length of the packet is larger than the remaining space in RAM, then the packet is dropped; (3) normal packets are written to RAM and the write pointer is updated. When the whole packet is written to RAM, the not empty flag is output, so that the FIFO data can be read [3].

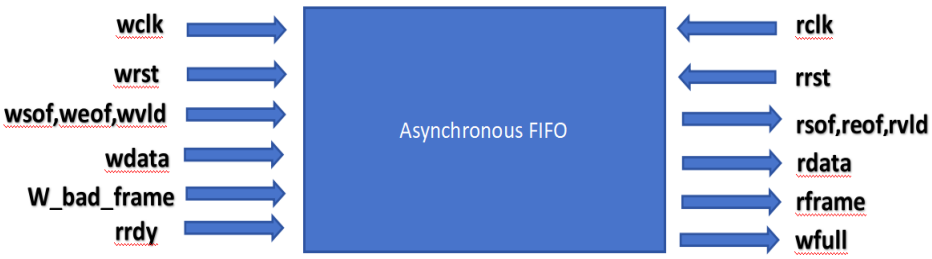

Fig 5. Signal interface for asynchronous FIFOs.

The circuit realizes access to the whole frame data of the Internet, and the software simulation and hardware implementation have been verified and have been practical applications. Practice has proved that it handles the whole frame data of the Internet with high reliability and stability. The design can be directly applied to the transmission and processing of Internet data and has a strong practical value in engineering.

5. Asynchronous Design for Parametric Design Problem Solving

The H. Ashour design implements parameterization of the data interface width and memory depth. Firstly, the design based on the read pointer and write pointer changing sequentially to form the null-full signal reflects the asynchronous FIFO design technique, so this also requires that both read and write pointers are parameterizable including the null-full signal to accommodate this.

The design solution is parameterizable through Verilog HDL code, where each module is individually configured and the code is independent so that each module in the architecture has an independent and self-contained functionality, providing a modular interface to other system-level components that can be reused in other systems. FIFO flag thresholds can be reconfigured at runtime, and these HDL parameters can be assigned during FIFO instantiation.

The parameterized design makes it possible to flexibly adjust the data interface width and memory depth according to different application scenarios, which greatly increases the versatility and adaptability of the design; at the same time, the modules in the architecture are functionally

independent, which is easy to maintain and expand, and can improve the reusability of the design and reduce the development cost.

Meanwhile, the FIFO flag threshold can be reconfigured at runtime, making it possible to meet the different requirements of different applications for the empty-full flag, which is suitable for application scenarios such as rate matching, reconfigurable hardware and data flow, as shown in Table 1 [4].

However, parameterized and modular design may increase the complexity of the design, for example, reconfiguring the flag threshold at runtime may bring some performance overhead and affect the read/write speed of the FIFO. Meanwhile, modular design may occupy more hardware resources, especially on FPGA platforms with limited resources, which may affect the implementation of other functional modules.

Table 1. Parameters.

Parameter	Description	Notes
DATA-SIZE	Determines the width of the data/data ports	
ADDR-SIZE	Determines the width of the write pointer, read pointer, near full threshold value and empty threshold value	FIFO memory depth is calculated to be two to the power of the value of this parameter

6. Conclusion

In this paper, different aspects of asynchronous FIFO design are studied in depth, and it is found that the use of grey code pointers and two-stage synchronizers can reduce the probability of sustainability and improve system stability. Although sub-stability may occur during jumps, it does not affect functionality and has significant value in cross-clock domain design.

Parallel loop structure reduces time and power consumption, providing low-latency, high-throughput performance without increasing latency as the number of FIFO stages increases, meeting high-performance requirements.

To solve the problem of caching whole frame data by FIFO, an asynchronous FIFO based on Verilog HDL high-level programmable language is designed, which can be easily used for intermediate caching of Internet data frames to ensure that Internet data frames are not easy to lose under high throughput.

Parametric and modular designs can increase versatility, adaptability and reusability and reduce development costs. However, they may increase complexity, affect read and write speeds, and take up more hardware resources.

Reference

[1] Wang Qishuang, Huang Zhenchun, PuHaifeng" FPGA-based asynchronous FIFO design and performance", Journal of Projectiles, Rockets, Missiles and Guidance, 2014, 34(06).

[2] Y. Xiao and R. Zhou, "Low latency high throughout circular asynchronous FIFO," in Tsinghua Science and Technology, Dec. 2008, vol. 13, no. 6, pp. 812-816.

[3] Shui Ying "FPGA-based frame-level asynchronous FIFO design" in Acoustics and Electronics Engineering, 2020(02).

[4] H. Ashour, "Design, simulation and realization of a parametrizable, configurable and modular asynchronous FIFO," 2015 Science and Information Conference (SAI), London, UK, 2015, pp. 1391-1395.

[5] Chen Xiaojun, Zhou, Guoxiang Asynchronous FIFO design with arbitrary depth in "Journal of Hefei University" 2011, 21(03).