ISSN 2959-6157

# Design and Implementation of a Width-Adjustable Asynchronous FIFO for Cross-Clock Domain Data Transmission

**Suyu Cheng** *

Department of Future Science and Engineering, Soochow University, Suzhou, 215222, China
*2128411001@stu.suda.edu.cn

**Abstract:**

Existing research shows that asynchronous FIFO has been widely used in a variety of complex system designs, such as FPGA-based image processing systems, USB communication systems, and so on. To make the designed asynchronous FIFO more programmable and flexible, so that it can adapt to various application scenarios more quickly, this paper studies and designs an asynchronous FIFO with adjustable input and output bit width. In this study, according to the relationship between 24-bit input data and 32-bit storage space, four 24-bit data are selected to fill three RAM storage spaces at one time. According to the multiple relationships between 32-bit storage space and 128-bit output data, the reading pointer is enlarged by 4 times to realize the function of reading four 32-bit data at one time. This whole process realizes the conversion of data bit width through data splicing. Finally, the 24-bit RGB888 image data is successfully converted into 128-bit data and transmitted to the bus, which successfully solves the problem of data width mismatch.

**Keywords:** Asynchronous fifo; Metastable state; Gray code.

## 1. Introduction

Existing research shows that asynchronous FIFO has been widely used in a variety of complex system designs, such as FPGA-based image processing systems, USB communication systems, and so on. To make the designed asynchronous FIFO more programmable and flexible, so that it can adapt to various application scenarios more quickly, this paper studies and designs an asynchronous FIFO with adjustable input and output bit width. In this study, according to the relationship between 24-bit input data and 32-bit storage space, four 24-bit data are selected to fill three RAM storage spaces at one time. According to the multiple relationships between 32-bit storage space and 128-bit output data, the reading pointer is enlarged by 4 times to realize the function of reading four 32-bit data at one time. This whole process realizes the conversion of data bit width through data splicing. Finally, the 24-bit RGB888 image data is successfully converted into 128-bit data and transmitted to the bus, which successfully solves the problem of data width mismatch.

## 2. Research Background

Asynchronous FIFO ( First In First Out ) is a first-in, first-out data buffer. Its read-and-write clocks are independent of each other, which is different from ordinary memory. It has no external read-and-write address lines. It can only read and write data in a certain order, and manage the data storage location through read and write pointers. In the process of data transmission across clock domains, due to the differences in frequency and phase between different clock domains, direct data transmission may lead to data loss. The use of asynchronous FIFO can effectively isolate different clock domains, to ensure the correct data transmission across clock domains. Because of the above characteristics, asynchronous FIFO is widely used in network interfaces, image processing, and other fields. In 1991, under the background of the increasing demand for data transmission between different clock domains in digital systems, Clifford E. Cummings proposed the concept of asynchronous FIFO. By using Gray code to convert binary pointers, the empty and full conditions can be directly judged. These studies still provide valuable theoretical guidance for modern cross-clock domain design, making asynchronous FIFO an important tool in digital system design. Next, I will first analyze the existing asynchronous FIFO research, and then analyze the asynchronous FIFO structure with adjustable input and output bit width designed by me, and then explain its application scenarios and detailed workflow. Finally, the advantages and future directions of this design are analyzed.

## 3. Literature Review

At present, there has been a lot of research based on asyn-

chronous FIFO. Aiming at the problem of output skew caused by unaligned data sources when multiple chips work at the same time, an asynchronous FIFO circuit with controllable delay is designed. The delay control module is added while realizing data transmission across the clock domain. The integer delay control is realized by adjusting the difference between the read pointer and the write pointer, and the high-precision decimal delay control is realized by adjusting the phase difference between the read clock and the write clock [1]. In the aerospace field, due to the existence of high-energy particles at high altitudes, the asynchronous FIFO will produce single-particle flipping, which will cause dysfunction. A new triple-module redundancy scheme can be used to quickly correct pointer errors and synchronize three redundant data, so as to achieve better single-particle protection effect [2] ; in order to meet the growing demand for fast access to Ethernet, the researchers use 0.18μm CMOS process design, the use of dual-port 8T structure to replace the memory, the use of latch amplifier and pre-charge technology to amplify the small signal on the bit line to design a high-speed asynchronous FIFO suitable for 100Gbit / s Ethernet PCS [3] ; In order to solve the problem of FIFO caching the whole frame data, the read and write enable of FIFO is built according to the BRAM block, and the generation and verification of read and write signals and empty and full signals are controlled to ensure that the Internet data frame is not lost in the case of large throughput [4] ; for the 32-bit floating-point DSP project, in order to match the peripherals and the CPU operating frequency to avoid the CPU in the waiting state and waste of resources, the researchers chose to use asynchronous FIFO to match the speed difference between the low-speed peripherals and the high-speed CPU [5]. It can be seen from the above applications that FIFO plays a very important role in various fields such as matching peripherals with different speeds, improving data transmission efficiency, controlling data transmission delay, caching, and data buffering. It is an indispensable key component in modern digital circuit design.

Gray code has always played a crucial role in the design of asynchronous FIFO. In the design of asynchronous FIFO based on Gray code, the generation of flag signal is accurately designed by comparing the read and write addresses to generate the correct empty and full flags and then synchronize to the corresponding clock domain. The use of the Gray code can effectively reduce the probability of a metastable state. In addition, the Gray code is compared with the single-step cyclic code. It is found that the use of Gray code can improve the stability of the system, but it will inevitably require more logic units, resulting in more transmission delay [6]. In addition, some researchers use Gray code as the pointer of asynchronous FIFO and compare it with the scheme of traditional binary code to represent the pointer. It is found that Gray code can effectively suppress the metastable state and make the performance of asynchronous FIFO more stable [7]. More researchers have proposed four design schemes based on Gray code, shift code, state flag, and interval address for the two key difficulties of metastable phenomenon and empty state judgment in asynchronous FIFO design. After comparison, it is found that the number of logic devices used in different design schemes is quite different. Among them, the number of devices used in asynchronous FIFO based on Gray code is the least, and the metastable phenomenon has also been effectively suppressed [8]. Combined with the above research, it can be seen that the method of using Gray code to represent the pointer has an effective inhibitory effect on the metastable state, and the required logic device is significantly reduced compared with the shift code. At the same time, because there is only 1 bit change between adjacent Gray codes, the multi-bit address is converted into Gray code, which greatly simplifies the processing across the clock domain. Although the advantages of asynchronous FIFO based on Gray code are very obvious, its shortcomings can not be ignored. In some cases, additional logic is required to convert and synchronize the Gray code, resulting in a slower average data transmission rate. When the depth of the FIFO is not a power of 2, the continuity of the Gray code is destroyed. Although this problem can be solved, it still brings a waste of resources. In addition, the complexity of the design and implementation of the Gray code itself cannot be ignored.

Metastability means that under certain conditions, the system is in a state between a stable state and an unstable state, and its duration cannot be determined. In digital circuits, the metastable state usually means that a trigger cannot reach a certain state within a specified time. Studies have shown that the influence of the metastable state on the system is mainly manifested in two aspects. One is that in the multi-fanout circuit, due to the difference in propagation delay, the load end identifies different logical levels, resulting in logical misjudgment; the other is the case where the output voltage is between the logic levels 0 and 1 [9]. Since the metastable state cannot be avoided, how to alleviate the metastable state has become the key to research. Since the cross-clock domain is often involved in FPGA design, researchers propose a cross-clock domain signal synchronization method based on trigger cascade for single bit level signal, pulse signal, and edge signal. For parallel signals, a cross-clock domain synchronization method combining asynchronous FIFO and handshake protocol is proposed, which successfully realizes cross-clock domain signal synchronization and greatly

alleviates the generation of metastability [10]. In addition to the metastability processing across the clock domain, researchers have explored the single clock domain metastability processing. Under the condition of a single clock domain, only the establishment / hold time margin of the trigger needs to be judged. If the establishment time of the subsequent trigger is insufficient, the trigger can be inserted between the two triggers to solve the problem [11].

## 4. Methods Technical Model Basis

The asynchronous FIFO structure designed in this study is shown in Figure 1. The numbers in the middle brackets in the picture represent the bit width of the data. It can be seen from the figure that the entire asynchronous FIFO structure (asyc_fifo) can be roughly divided into three parts. The first part is the data conversion module 1, whose function is to convert the input data din with a bit width of 24 into 32-bit write data. The second part is a 32-bit dual-port RAM located in the middle of the image. The third part is the data conversion module 2, which is located on the right side of the picture. Its main function is to convert the 32-bit data stored in the dual-port RAM into 128-bit output data; in addition to the input data and output data, many other signals can be seen from the figure. For example, read clock signal din_clk, write clock signal dout_clk, write enable signal din_en, read enable signal dout_en, read empty signal empty, write full signal full and an additional programmable full signal prog _ full. The read clock signal dout_clk and the write clock signal din_clk determine the clock frequency of the read and write operation, respectively. The read-enabled signal dout_en and the write-enabled signal din_en determine whether to perform the read and write operation. Only when the write-enabled signal or read-enabled signal is 1, the asynchronous FIFO will perform the read or write operation. When the storage space in the dual-port RAM is insufficient, it will make the write signal output high level. When the data in the dual-port RAM cannot meet the needs of a read operation, it will make the read empty signal output low level. It can be seen from the figure that in addition to the asyc_fifo module located in the center, there is a Vcam module and bus module on the left and right sides respectively. The Vcam module is a virtual camera, which is responsible for providing 24-bit image data as input data into asynchronous FIFO. The bus module represents a bus with a bit width of 128 bits. Its main function is to receive the output signal and transmit it to the host computer.

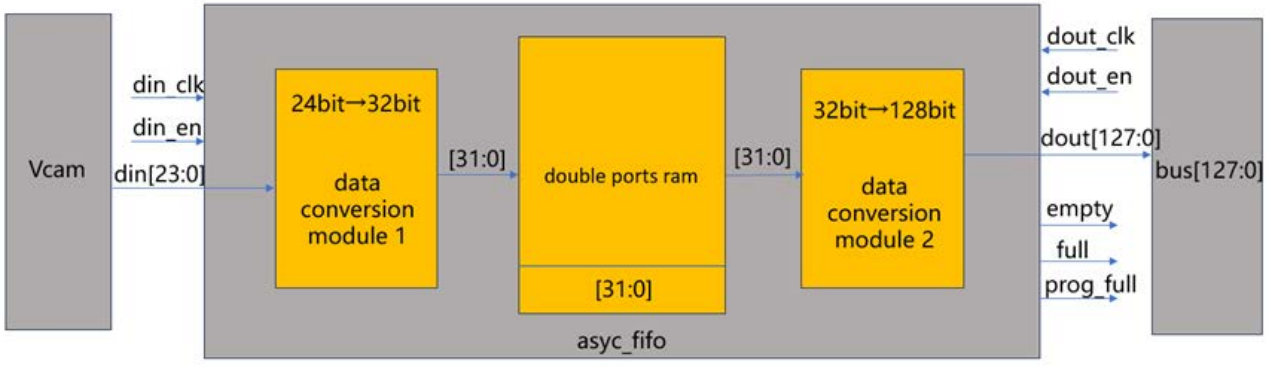
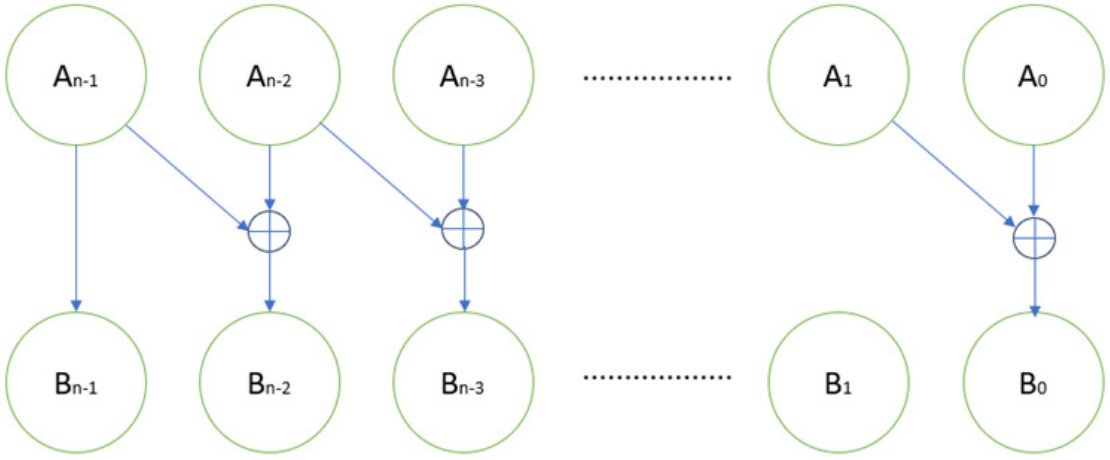



**Fig. 1 Input and output adjustable asynchronous FIFO structure**

Picture credit : Original

Gray code is a special binary coding method. Its main feature is that there is only one bit of binary number between two adjacent Gray codes, and there is only one bit of binary number between its maximum number and minimum number. Gray code is a kind of unweighted code. Each bit of it has no fixed weight, so it will be more reliable than BCD code in some specific situations. The encoding rules of the binary code are shown in Figure 2. From the diagram, it can be seen that the highest bit of Gray code remains unchanged compared with its original corresponding binary code, while the second high bit is the result of the XOR operation between the highest bit and the second high bit of the binary code corresponding to the Gray code. It is precise because of the characteristic that there is only a one-bit difference between two adjacent numbers. When the multi-bit signal is transmitted across the clock domain, after the pointer is represented by the Gray code, even if the metastable state occurs, only one bit will be affected. Only two possibilities are returning to the previous state or correctly entering the next state. Even if returning to the previous state, it will only bring a

certain delay without affecting its function. In summary, Gray code significantly reduces the occurrence probability of metastability through its single-bit variation characteristics.




| $A_2A_1A_0$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| $B_2B_1B_0$ | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

**Fig. 2 Gray code coding rule diagram**

Picture credit : Original

Since the read and write addresses in the asynchronous FIFO are controlled by different clocks, it is impossible to directly compare the two addresses. Therefore, we often need to synchronize the read and write addresses in the asynchronous FIFO. One of the most commonly used methods for synchronizing read and write addresses is to synchronize the write address from the write clock domain to an intermediate clock domain, and then synchronize the intermediate clock domain to the read clock domain so that the synchronization of the write clock domain to the read clock domain is realized, which in turn synchronizes the read clock domain to the write clock domain. The specific synchronization process is shown in Fig.3.The read-and-write clock domain can be roughly divided into two cases. One is that the read clock domain is faster than the write clock domain. In this case, the read address is compared with the write address after the two-stage D flip-flop. During the synchronization process, the D flip-flop has a delay, resulting in a time lag. Therefore, the read address we compare is smaller than the current actual read address. At this time, comparing the smaller read address with the write address will get a full signal, but at this time, the FIFO may not be written full. The process is conservative, so there is no overwrite phenomenon in the FIFO; The other is the case that the read clock domain is slower than the write clock domain. In this case, the write address is compared with the read address after passing through the two-level D flip-flop. The D flip-flop has a delay during the synchronization process, resulting in a time lag. Therefore, the write address we compare is smaller than the current actual write address. At this time, the smaller write address is compared with the read address, and an empty signal is obtained. However, the FIFO may not be read empty at this time. The process is conservative, so the FIFO does not have an over-read phenomenon. Combining the above two cases, it can be found that although the two-stage register consumes time and reduces the performance of FIFO to a certain extent, it does not affect the function. Therefore, the two-stage register can be used for synchronization in both cases.

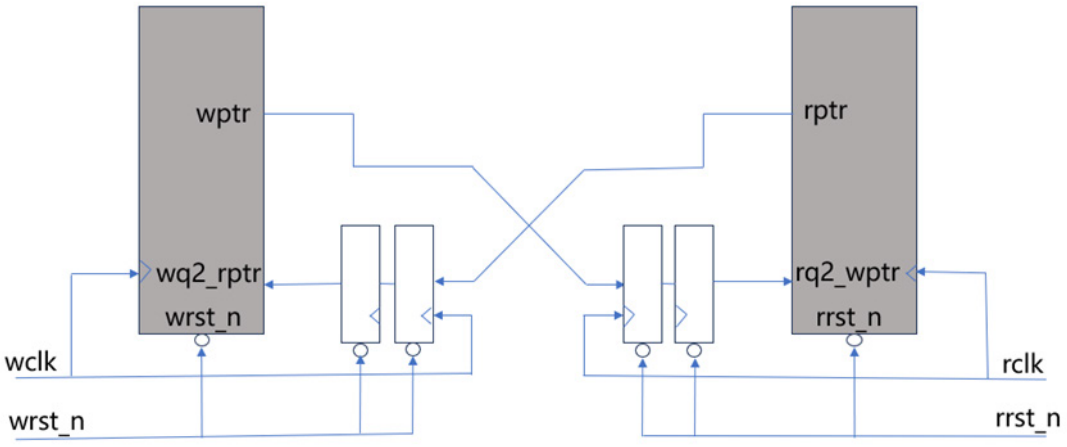



**Fig. 3 Reading and writing pointer synchronization**

Picture credit : Original

In this research design, the input device I selected is a virtual camera, which will generate RGB888 image data. The bit width of this data is 24 bits. In the experiment, the storage space size of the RAM I selected is 32 bits. The data read from the RAM will be transmitted to an Avalon bus with a data bit width of 128 bits, and finally the data will be transmitted to the host computer by the bus. In this process, the input signal is 24 bits and the size of a storage space in RAM is 32 bits. Therefore, the first data conversion is required to convert the 24-bit RGB888 image data into a 32-bit signal and store it in the RAM storage space with maximum efficiency. Because the data bit width of the bus is 128 bits, a second data conversion is required to convert the 32-bit stored data into 128-bit data and output it to the bus. In this process, to achieve 24-bit to 32-bit data conversion, I choose to read four 24-bit image data at one time and store them in three storage spaces of RAM. The specific data conversion process is shown in Figure 4, to maximize the utilization of RAM storage space; to convert the 32-bit stored data into 128-bit data and output it to the bus, 128 is 4 times 32, so I choose to expand the read pointer synchronized to the write clock domain to 4 times of the original, to realize the function of reading 4 32-bit stored data and realize the conversion from 32-bit to 128-bit. The specific read pointer amplification process is shown in Figure 5.

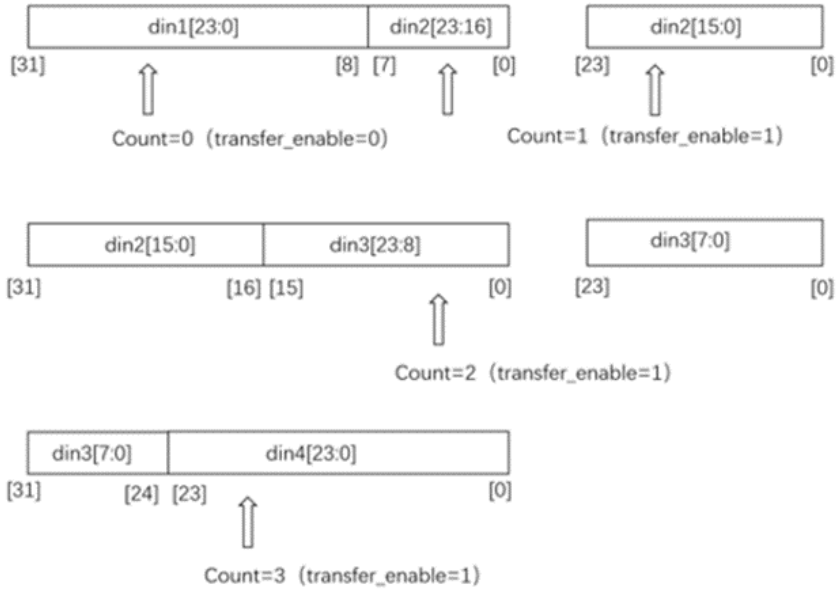



**Fig. 4 24-bit to 32-bit process**

Picture credit : Original

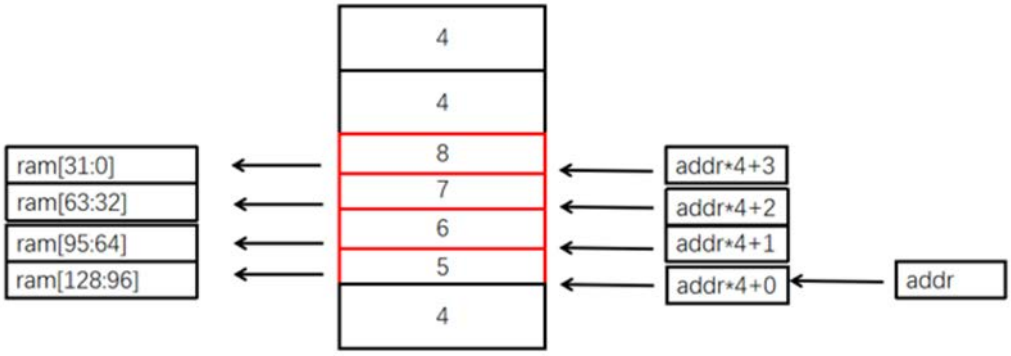



**Fig. 5 Reading pointer expansion**

Picture credit : Original

## 5. Experimental and Model Evaluation

The asynchronous FIFO design with adjustable input and output bit width mainly realizes the function of converting 24-bit RGB888 image data into 128-bit data output. This idea is based on a system that contains a video camera module inside. The video camera module is Vcam. The main function of this module is to capture the image and output one of its RGB888 image data formats. Since the clock frequency of the Vcam may not be synchronized with the clocks of other modules such as processor modules and display devices in the system, it will involve cross-clock domain data transmission. Therefore, in this experiment, asynchronous FIFO is selected to buffer and synchronize the data transmitted across the clock domain. These image data will be stored in dual-port RAM after the first data conversion, and data loss or disorder will be avoided through asynchronous FIFO read-and-write control. When the data is ready, RAM storage space will be filled, and then the asynchronous FIFO reads 128 bits of data and transmits it to the Avalon bus, and then by the bus transmission to other modules for use. Here, because the Avalon bus will perform burst transmission with a burst length of 16, it is necessary to prepare 16 128-bit data in the FIFO before writing the data into the subsequent modules at one time. Therefore, I designed a programmable full signal prog_full in the FIFO. The judgment value of this signal is set to 64, because the sum of 16 128-bit data is equal to the sum of 64 32-bit storage data, to determine whether the bus can perform burst transmission.

After the waveform simulation, I generally divide the waveform into two parts. One is the initialization process. In the initialization process, first, define a writing task, set din as a random number, and set din_en to 1. When the write clock falling edge arrives, the write enable is set to 0; a reading task is defined, and dout_en is set to 1. When the write clock falling edge arrives, the write enable is set to 0. A register type 4-bit counter is defined, and an always module is defined. When the rising edge of the write clock comes, the random number is imported into the counter. Then the set of asynchronous fifo programs is instantiated; Two variables i and j are defined, and each data is initialized and assigned by the initial function, and then a for loop is used for i. The condition set here is i < 200 instead of 128. This is because the data we need has 128 * 32 bits, and the input data is only 24 bits at a time. If only 128 cycles cannot fill the internal space of RAM, it is necessary to take a number larger than 128 * 32 / 24. 200 is to meet this condition. In the for loop, the write clock rising edge is selected as the sensitive condition, the write enable is set to 1, and the write data is set to random, to achieve the purpose of randomly filling the RAM space, and then a flag signal is defined as the mark of RAM filling. When flag = 1, if the counter count is less than or equal to 8, the write operation is performed, otherwise the read operation is performed. Before the RAM is not filled, the read operation is unable to read the stable value, so the dout in the waveform diagram has been in an unknown state, which is also the most significant waveform feature in the initialization process. The specific waveform of the initialization process is shown in Figure 6. Another part of the waveform is the execution process of random read and write. It can be seen from the figure that the designed write enable is randomly raised at the rising edge of the write clock, and the write data changes with the rise of the write enable. The designed read enable is always raised, and the read data changes with the rise of the read clock. It can also be seen from the figure that the write data is 24-bit and the read data is 128-bit, which successfully realizes the change of input and output bit width. The specific waveform of the random read-and-write process is shown in Figure 7.

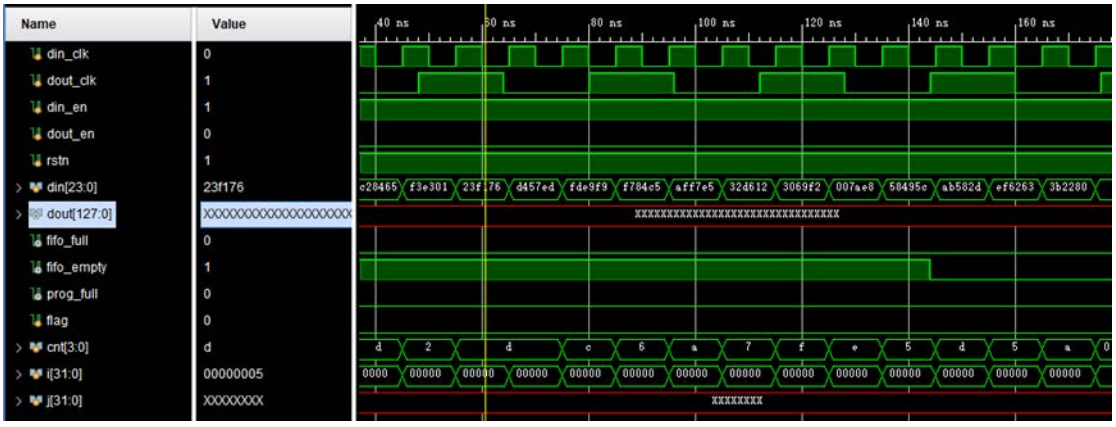



**Fig. 6 Initialization process**

Picture credit : Original

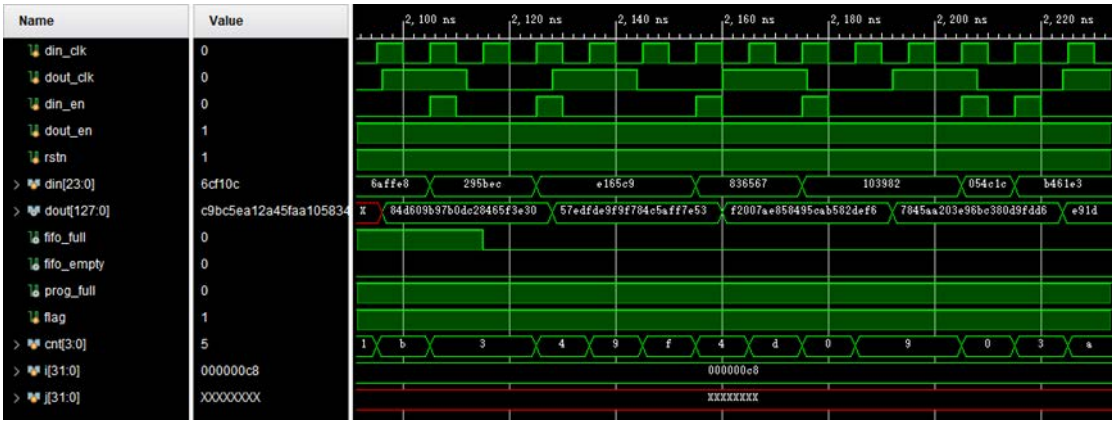



**Fig. 7 Random read-write process**

Picture credit : Original

## 6. Conclusion

The input and output bit width adjustable asynchronous FIFO design can be effectively applied to many systems involving different bit width data. The adjustability of its input and output bit width makes it adapt to different data transmission requirements. It can not only convert small bit width into large bit width data through similar methods but also provide ideas and methods for the transmission of large bit width data into small bit width data. Because this design is based on asynchronous FIFO, it can also be applied to multiple processor systems and multiple network interfaces. The scenario provides conditions for cross-clock domain transmission of data and avoids data loss caused by clock asynchrony. In the design process, the instantiation program is adopted, which can flexibly adjust the depth and width of FIFO according to the change of demand, so that the system has stronger adjustability and adaptability, and can quickly adapt to different application scenarios.

To improve performance and resource utilization, we can consider designing a more efficient storage structure than dual-port RAM in the future, such as distributed LUT. In the existing design, more functional modules with different uses can be considered to improve the overall performance of the design. For example, a metastable check module can be introduced to ensure the accuracy and reliability of the data transmission process; in this design, the Gray code is finally decoded into binary code to judge the empty and full signals. In the future, we can continue to explore the rules of using Gray code to judge the empty and full signals. Through more in-depth research on the above points, we can further improve the overall performance and utilization range of the input and output bit width adjustable asynchronous FIFO design, and provide a more reliable and efficient solution for data transmission of complex systems.

## References

[1] Chen Tingting, Lu Feng, Wan Shuqin, et al. Design of a Controllable Delay Asynchronous FIFO Circuit [J]. Microelectronics, 2022,52 (1) : 42-46. DOI : 10.13911 / j.cnki.1004-3365.210280.

[2] Sun Yuan, Ren Yiqun, Fan Yuyang. Design of AFIFO anti-SEU [J].Modern electronic technology, 2023,46 (11) : 160-164. DOI : 10.16652 / j.issn.1004-373x.2023.11.029.
[3] Zhan Yongzheng, Li Tuo, Hu Qingsheng, et al. A High-Speed Asynchronous FIFO for 100 Gbit/s Ethernet PCS [J]. Microelectronics, 2022, 52 (5) : 886-892. DOI : 10.13911 / j.cnki.1004-3365.210404.
[4] Shui Ying. Frame-level asynchronous FIFO design based on FPGA [J].Acoustic and electronic engineering, 2020 (2) : 32-34.
[5] Wu Xiuying, Huang Songren. Research and design of asynchronous FIFO in floating-point DSP [J]. Electronic World, 2018,0 (1) : 145-146.
[6] Cong Hongyan, Liu Ying, Wan Qing. Optimization and Implementation Based Gray Code and Depth Configurable Asynchronous FIFO [J].Electronics and Packaging, 2014 (5) : 33-36.
[7] Wu Kun, Huang Kun, Fu Yong, et al. A Design and Realization of Asynchronous FIFO Based on Gray Code [J]. Computer and Digital Engineering, 2007,35 (01) : 141-144.
[8] Wang Qishuang, Huang Zhenchun, Pu Haifeng. Designs and Performance Study for Asynchronous FIFO Based on FPGA [J]. Journal of Missiles and Rockets and Guidance, 2014,34 (6) : 185-189. DOI : 10.3969 / j.issn.1673-9728.2014.047.
[9] Wang Luyuan. Metastable state and mitigation measures in FPGA design [J]. Electronic Technology Applications, 2012,38 (8) : 13-15,19. DOI : 10.3969 / j.issn.0258-7998.2012.08.004.
[10] Yang Yanyan, Si Qianran, Ma Xianying et al. Research on Metastability and Its Mitigation Methods in FPGA Design [J]. Journal of Aircraft Measurement and Control, 2014,33 (03) : 208-213.
[11] Zhu Yu, Dong Guantao, Zhang Shuo.Research on metastable processing technology of FPGA software [J].China Inspection and Testing, 2020,28 (3) : 14-17. DOI : 10.16428 / j.cnki.cn10-1469 / tb.2020.03.004.