Journal of Engineering and Applied Sciences 14 (1): 159-163, 2019
ISSN: 1816-949X
© Medwell Journals, 2019

Design and Verification of Asynchronous FIFO with Novel Architecture Using Verilog HDL

Avinash Yadlapati and Hari Kishore Kakarla
Department of Electrical and Computer Engineering (ECE), KL University Green Fields, 522502, Vaddeswaram, Andhra Pradesh (AP), India

Abstract: A FIFO is a “First In First Out” memory queue between any two asynchronous domains with simultaneous write and read access to and from the FIFO, these accesses being on different clocks. The FIFO has input ports like data input (write), write clock, read clock, reset and output ports like FIFO full flag, data out (read) and FIFO empty flag. It also has control signals like write enable and read enable. The most important signals that control the FIFO operation are the write pointer and the read pointer. These pointers in the case of Synchronous FIFO operate in a single clock while in the case of Asynchronous FIFO operate in two clocks, write clock and read clock respectively. FIFO can be either Synchronous or Asynchronous. The basic difference between them is that the entire operation of Synchronous FIFO is entirely dependent on the clock whereas the write operation and read operation of Asynchronous FIFO are asynchronous to each other. In this study a Novel approach to designing an Asynchronous FIFO is used. Instead of taking a separate bit to identify whether the FIFO is full or empty, the resaerchers have used an internal signal (last operation) to identify if the FIFO is full or empty.

Key words: Asynchronous FIFO, write pointer, read pointer, flags, synchronization, gray code converter, FIFO full, FIFO empty, memory queue

INTRODUCTION

FIFO stands for First-In First-Out memory buffer and it is used to transfer the data between two different clock domains with the technique of synchronization (Anonymous, 2004). This study talks about the design and implementation of an Asynchronous FIFO with novel architecture in which the synchronization is done using a gray code mechanism and by generating an internal signal last operation. FIFO’s are primarily of two types:

- Synchronous FIFO
- Asynchronous FIFO

In a Synchronous FIFO a single clock is used for both writing and reading data while in Asynchronous FIFO two separate clocks are used to read and write data which means that both write and read are independent of each other and hence, the mechanism of read and write is faster compared with Synchronous FIFO (Cummings and Alfke, 2003), i.e., two clocks are required for read and write osperations. Each clock has separate frequency. Every FIFO has a different architecture but this study details novel architecture that uses a signal “last operation) to generate the FIFO full and FIFO empty flags and a gray code synchronization in order to reduce the switching activities thereby reducing the power dissipation in the circuit (Han and Stevens, 2009).

MATERIALS AND METHODS

Block diagram of Asynchronous FIFO: Figure 1 shows the diagrammatic representation of an Asynchronous FIFO. The input ports for an Asynchronous FIFO are as follows:

- wr_clk
- rd_clk
- wr_en
- rd_en
- rst
- data_in

The output ports for an Asynchronous FIFO are as follows:

- data_out
- FIFO_full
- FIFO_empty

Corresponding Author: Avinash Yadlapati, Department of Electrical and Computer Engineering (ECE), KL University Green Fields, 522502, Vaddeswaram, Andhra Pradesh (AP), India

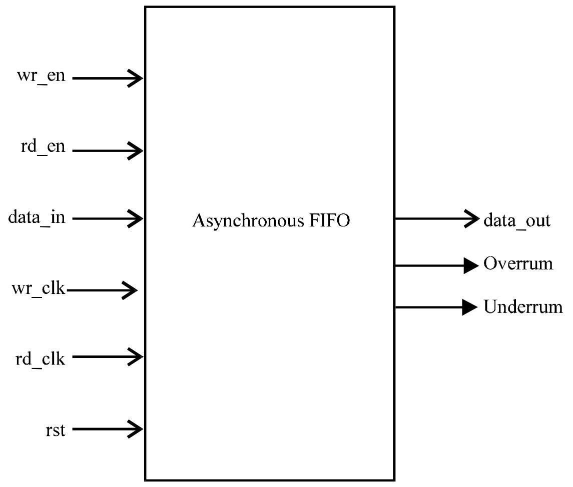

Fig. 1: Block diagram of Asynchronous FIFO

We will use separate clocks for read and write operation, wr_clk and rd_clk are the input clocks that are used for write and read mechanisms accordingly. The signal rst is for resetting the data in the Asynchronous FIFO and to bring the FIFO to a known state. It is very dangerous for the memory to not have a reset signal as it will be filled with junk data values in the absence of rst signal. data_in is the input data to the FIFO. wr_en and rd_en are the control signals to the FIFO and the complete write and read operation happens at the behest of these control signals. FIFO_full and FIFO_empty are the status flags which will tell the processor or the master when the FIFO is full or when the FIFO is empty (Ramesh *et al*., 2012).

Novel architecture of Asynchronous FIFO: This study details a unique architecture which differentiates this FIFO design from the other designs of similar application. Some excellent work has already been published on the design and implementation of Asynchronous FiFO's. Yet, as a unique architecture with constraints like area, power and throughput are implemented in this study. As a good design, every RTL design should be designed using multiplexers, D-flip-flops and the design should be made as modular as possible. This study discusses the architecture that is designed only with the help of mux's and flip-flops. The primary concept of this architecture is the last operation mechanism which differentiates it from other architectures (Fig. 2).

Managing pointers in Asynchronous FIFO: The complete FIFO operation is controlled by the write pointer and the read pointer. Initially they are both pointing to the zeroth location which means that the FIFO is in reset state. The pointers will be incremented by the control signals write enable and the read enable. When the write enable is high, the write pointer is incremented at every edge of the write clock and when the read enable is high, the read pointer is incremented at every rising edge of the read clock. Once the pointer is incremented, it means that data is written in the previous location. For instance, when the write pointer is initially pointing to zero location and the write enable signal is asserted, the write pointer gets incremented and now the current location of the write pointer is location 1. But the data is present in the zeroth location. Similarly, in the case of read pointer when the read enable signal is asserted, the read pointer increments to the next location which is location 1 but the read data comes from the zeroth location. If the number of address locations is "n", after addressing all the locations the pointers come to the initial state depending on the enable signals. If data is continuously written into memory and no read operation is performed then the write pointer will come to the initial state and will retain in the same state even if write enable signal is asserted, the same thing happens with the read pointer too.

In the case of Synchronous FIFO, the FIFO full flag is asserted when both the pointers are pointing to the same location and when the last operation is write and the FIFO empty flag is asserted when both the pointers point to the same location and the last operation is a read (Anonymous, 2004).

In Asynchronous FIFO design it is not possible because here two different and asynchronous clocks are used which would be essential to control the counter. So, in order to determine those conditions read and write pointers are to be compared along with individual previous operation for write and read as they operate at different frequencies.

Application of gray code converters: Gray code converter is used for reducing the switching activity and thereby reducing the probability of metastable condition. The other characteristic of gray code comes when representing successive binary numbers, reflecting only 1 bit change for each increment in binary values, thus, reducing the switching activity and hence, power (Anonymous, 2018). The lower switching activity also accomplishes less glitch formation, thus, reducing any metastability (Fig. 3).

The importance of reducing the metastable condition comes when comparing the two pointers. By reducing the, number of transitions the possibility of interpreting a signal transitioning from 10-0 or 0-1 as 1's and 0's respectively, by the combinational logic (xnor gate as an equivalency check) will become easier.

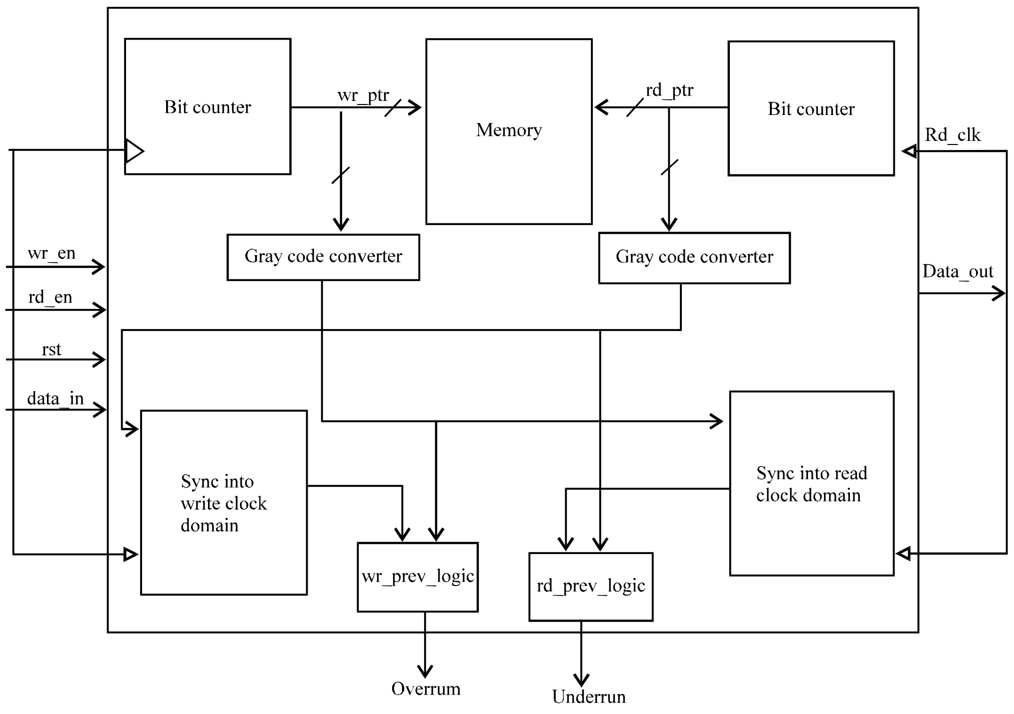

Fig. 2: Novel architecture of Asynchronous FIFO

100
101
111
110
010
011
001
000

Fig. 3: Gray code sequence

Importance of synchronizer: FIFO full condition is critical for write operation and FIFO empty condition is critical for read operation. Full and empty conditions are critical for write and read inhibition respectively. FIFO full and FIFO empty conditions are generated based on the write and

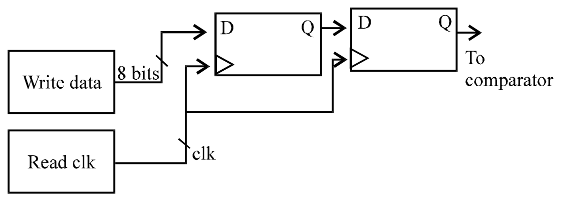

Fig. 4: Dual flop synchronization

read pointers position, respectively. Being an asynchronous design write and read mechanisms increment respective pointers with different clock frequencies. Synchronization is achieved by syncing the gray code write pointer with read clock domain and viz.

Hence, there's a need for synchronizing the gray code write pointer with read clock and gay code read pointer with write clock.

Figure 4 the synchronization mechanism. In general, a 1bit data is synchronized using two flip-flops. The write data is synchronized with the read clock and the read data will be synchronized with the write clock.

Generation of FIFO full and FIFO empty flags: The FIFO full flag and FIFO empty flag are intended to prevent FIFO full and FIFO empty conditions, i.e., the FIFO should not be allowed to write data again once it becomes full and the FIFO should not be allowed to read the data again once it becomes empty.

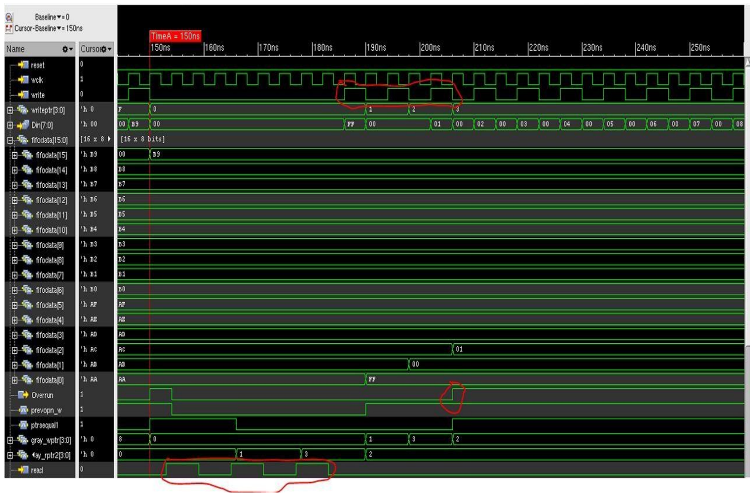

Fig. 5: FIFO full flag generation

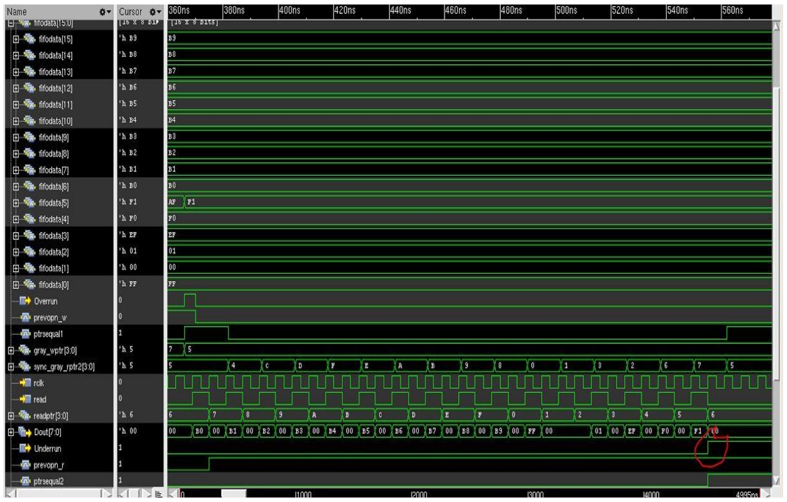

Fig. 6: FIFO empty flag generation

When FIFO full flag (when write pointer equals the synchronized read pointer and the last operation is write) is asserted the total memory locations are filled with data and there are no free locations to write any further data, hence, no write operation can be done. When there is FIFO empty flag (when read pointer equals synchronized write pointer and the last operation is a read) is asserted, the data is read out from all the memory locations and the FIFO is devoid of any new data, hence, no read operation can be done (Anonymous, 2009).

Asynchronous FIFO Implementation: The read or write operation is done according to the user request. When the user requests a write operation data is loaded into the memory and the location will be specified by the write pointer. When the user requests for a read operation the data that is loaded into the memory is read out. The read and write pointers keep on incrementing until it reaches the last location and again the pointers come to initial location which is also known as wrap up condition. This being a FIFO the first written data will be read out first and then from the next location and so on.

The memory specified in this study can address "n" locations of m-bit wide data. When the user requests for continuous write operation and no read operation the memory will load data until all the "n" locations are filled and then it remains in the same state, i.e., it preserves the last data (FIFO full condition), until read enable signal is asserted (Bergeron, 2003). If the memory is completely filled and the user requests for a read operation, the data will be read out until there remains no data in the memory. Even if the user requests a read operation, the FIFO asserts FIFO empty signal and the circuit preserves the previous state. When there occurs read and write operations simultaneously the data that is written into the memory first will be read out first. The pointers will look after the addressesto which memory location the data is to be sent and from which memory location the data is to be read from.

From Fig. 2, it's clear that the full and empty status flags are generated by previous operation logic and comparison of gray code pointers using a comparator after synchronization of the read pointer with write clock and viz. The counter used in this design is an n-bit counter and the comparison between the read and write gray code pointers is done as follows:

- If the pointers are equal and last operation is write, then FIFO full flag is asserted
- If the pointers are equal and last operation is read, then FIFO empty flag is asserted

RESULTS AND DISCUSSION

Simulations are carried out on Synopsys VCS using v erilog as a hardware description language. Random test cases are used to verify the Asynchronous FIFO functionality (Bergeron, 2003).

Figure 5 shows that initially FIFO empty flag is asserted as there's no data initially loaded into the memory. After write enable is asserted there is a continuous write operation and no read operation requested until the FIFO depth is completely filled. Hence, FIFO full flag is asserted. We can find that once the data is loaded into memory there is a de-assertion of FiFO empty flag. After read operation is requested we can observe that there's de-assertion of FiFO full flag. Simultaneous read and write operations are also depicted in Fig. 5. Hence, Asynchronous FiFO functionality is verified (Bhasker, 1998). Figure 6 shows that FiFO empty flag is asserted when the FiFO data is read out completely.

In this study, a unique strategy to implement a rapid Asynchronous FIFO using gray counter Synchronization and with the help of last operation is shown. In this design the gray counters uses a comparator along with the last operation for the generation of FIFO full and FIFO empty status flags. This being an asynchronous FIFO a lot of effort is required to design and meet the timing of the read clock with the write clock and viz. This novel architecture also overcomes the problem of metastability and mean time between failures by using Gray code counters. Continuous writing of data into memory, continuous reading of data from memory and simultaneous reading and writing of data from memory are verified with different test cases. All these test cases are verified using Synopsys VCS simulator.

CONCLUSION

This has reduced the hardware generated after Synthesis and also, the FIFO is designed in such a way that outputs are registered and no combinational glitches or spikes are encountered. The entire design is implemented using synthesizable verilog RTL code and verified using Synopsys VCS simulator.

ACKNOWLEDGEMENTS

I would like to thank the staff of KL University for their immense support by providing the necessary tools and infrastructure for carrying out the research work and motivation in writing this study. Without their guidance and support this study would not have seen the light of the day.

REFERENCES

Anonymous, 2004. Asynchronous FIFO v6.1. Xilinx, San Jose, California, USA. https://www.xilinx.com/support/documentation/ip_documentation/async_FIFO.pdf

Anonymous, 2009. Tutorials on System verilog, Verilog, Open Vera, Verification, OVM, VMM, AXI, OCP. ASIC Company, Sydney, Australia. http://www.asicguru.com/

Anonymous, 2018. Asynchronous FIFO in virtex-FPGA's. Australian Securities and Investments Commission, Sydney, Australia.

Bergeron, J., 2003. Writing Testbenches: Functional Verification of HDL Models. Springer, Berlin, Germany, ISBN:9781402074011, Pages: 475.

Bhasker, J., 1998. Verilog HDL Synthesis: A Practical Primer. Star Galaxy Publishing, Pennsylvania, USA., ISBN:9780965039154, Pages: 215.

Cummings, C.E. and P. Alfke, 2003. Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons. In: Verilog HDL: A Guide to Digital Design and Synthesis, Palnitkar, S. (Ed.). Sun Microsystems Press Publisher, California?, USA., pp: 1-18.

Han, H. and K.S. Stevens, 2009. Clocked and asynchronous FIFO characterization and comparison. Proceedings of the 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC'09), October 12-14, 2009, IEEE, Florianopolis, Brazil, ISBN:978-1-4577-0237-2, pp: 101-108.

Ramesh, G., V.S. Kumar and K.J. Reddy, 2012. Asynchronous FIFO design with gray code pointer for high speed AMBA AHB compliant memory controller. IOSR. J. VLSI Sig. Process., 1: 32-37.