I J C T A, 10(8), 2017, pp. 343-349 ISSN: 0974-5572

Novel Architecture for Designing Asynchronous First in First out (FIFO)

Avinash Yadlapati* and Hari Kishore Kakarla*

ABSTRACT

The Asynchronous FIFO is a first-in first-out memory queue that achieves synchronization of read clock domain and write clock domain with the help of handshaking signals, Overrun and Underrun conditions. This being an asynchronous FIFO is used to pipe the data from write domain to read domain working at different frequency. So the read and write pointers need to be synchronized, in order to generate the Overrun and Underrun status flags. This paper details the novel architecture for designing the Overrun and Underrun flag generation with the help of "Previous Operation", usage of gray code converters and synchronization mechanism with help of dual flop architecture. The FIFO architecture that is specified in this paper is implemented as configurable sub-system and it can be reused as an IP for any System-on-Chip (SoC) designs. The entire design is implemented using synthesizable Verilog RTL Code and verified with Cadence NC simulator.

Keyword: Asynchronous FIFO, Pointers, Status flags, Synchronization, Gray code converter, Overrun, Underrun

1. INTRODUCTION

As discussed earlier, FIFO stands for first-in first-out and it is used to transfer the data by achieving synchronization [1]. In this paper an asynchronous FIFO is designed with novel architecture in which the synchronization is done using a gray code mechanism. In general, we have two characterizations in FIFO and they are

1. Synchronous FIFO
2. Asynchronous FIFO

In a Synchronous FIFO a solitary clock is used for reading data from memory and for composing data into memory i.e. read and write operations are performed using a single clock while in Asynchronous FIFO a separate clock is utilized for reading data from memory and a separate clock for composing data into memory[2]. I.e. Two clocks are required for read and write operations. Each clock has a separate frequency. Generally as per the prerequisite of the passage of data the user chooses the frequency. Every FIFO has a different architecture, but this paper details novel architecture that uses a signal "previous_operation) to generate the Overrun and Underrun conditions and a gray code mechanism in synchronization in order to reduce the switching activities, as well as power consumption while preventing the design from undergoing metastable condition [2].

2. SCHEMATIC SYMBOL OF ASYNCHRONOUS FIFO

Figure 1 shows the schematic symbol of an Asynchronous FIFO. All the signals that were shown are the port level signals and the input signals are:

* Department of ECE, KL University Green Fields, Vaddeswaram-522502, A.P, India, *Emails: avinash.amd@gmail.com, kakarla.harikishore@kluniversity.in*

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

Output signals are:

- data_out
- Overrun
- Underrun

As mentioned earlier we use separate clocks for read and write operation, wr_clk and rd_clk are the clocks that are used for write and read operation accordingly. The signal reset is for resetting the data from all the storage elements and the data that is present in the memory is flushed out once a reset signal is given i.e. the whole data will be read from the memory and then it is taken to a known state. Data_in is the signal through which we are sending the data. FIFO_full and FIFO_empty are the status flags, based upon these signals the user will request either read or write operation [3].

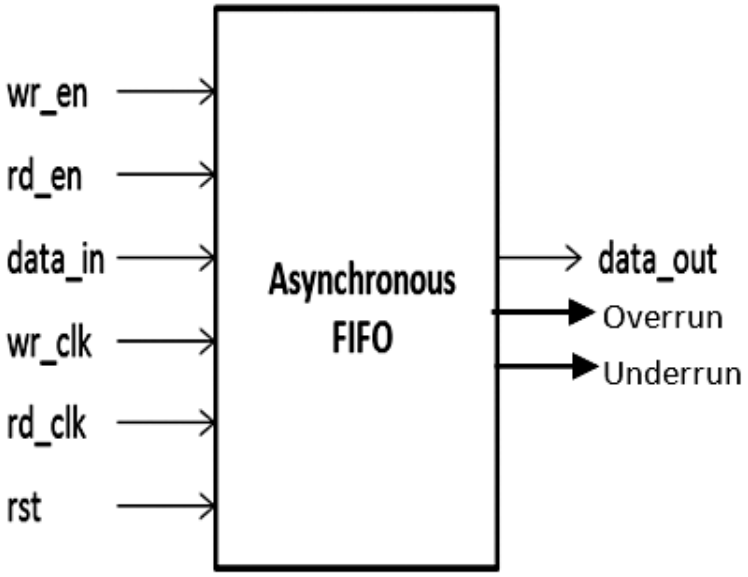

Figure 1: Asynchronous FIFO schematic symbol

3. ARCHITECTURE OF ASYNCHRONOUS FIFO

This paper details a versatile architecture which differentiates this FIFO design from the other designs. As a good design, every RTL design should be comprised of mux's and flip-flops. This paper discusses the architecture that is designed only with the help of mux's and flip-flops. The soul of this architecture is the last operation mechanism.

3.1. Read and Write Pointers of Asynchronous FIFO

- To understand FIFO design one should first study about the read and write pointers in a FIFO as the whole operation is dependent on these pointers itself. Instead of shifting the data that is to be read

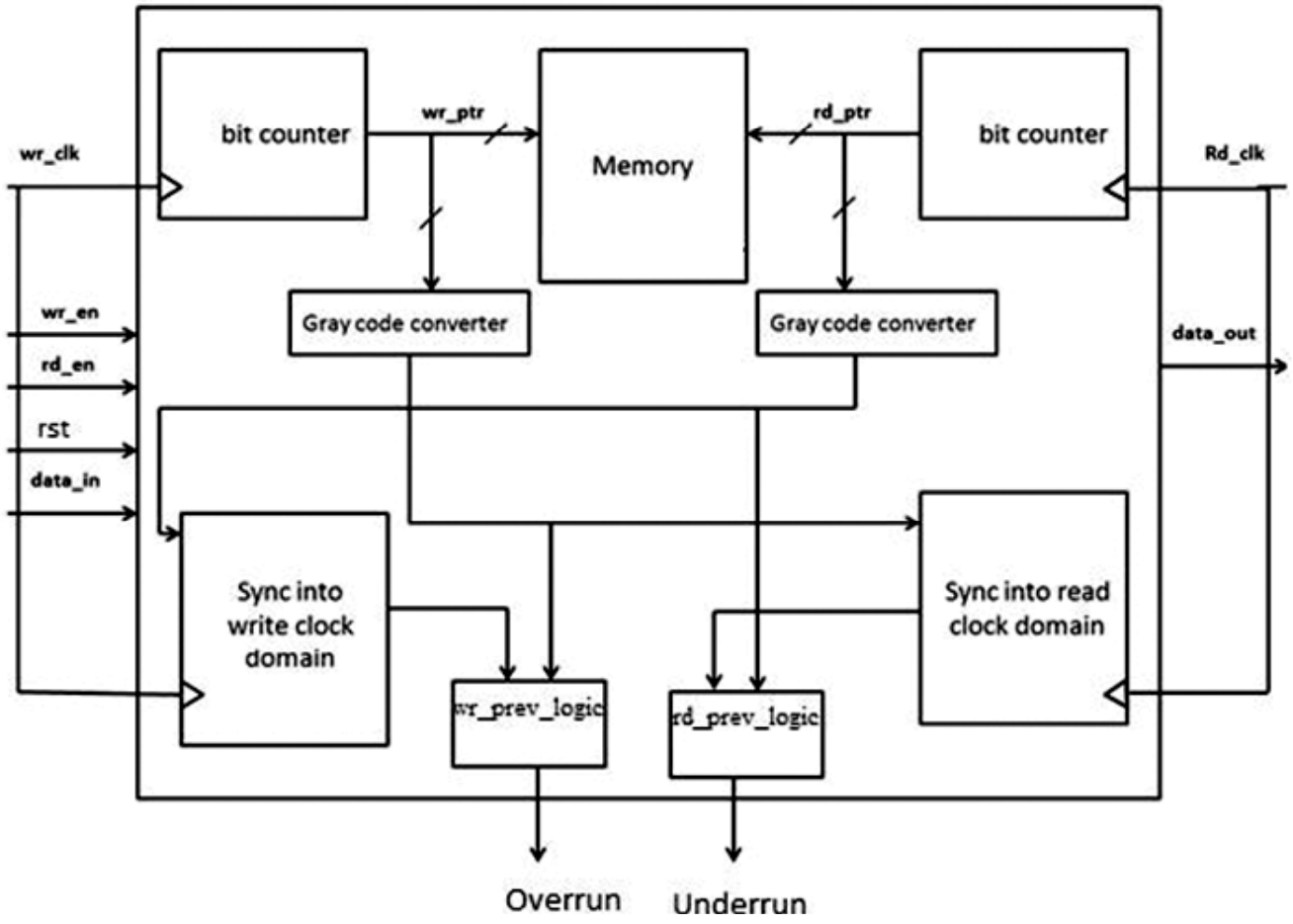

Figure 2: Architecture of Asynchronous FIFO

or written from one address location to another, it is better to consider pointers and shift them accordingly. So that a lot of circuitry and complexity will be reduced and this makes an efficient design. These pointers address the memory location. Initially the read and write pointers are initialized and then onwards according to the assertion of rd_en and wr_en signal the read and write pointers are incremented. After completion of read or write operation the read or write pointer will immediately points to the next address 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.

- For a synchronous FIFO the underrun and overrun flags can be generated by using a counter that counts the number of continuous writes, continuous writes and simultaneous read, write operations [4] along with the previous operation.
- 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.

3.2. Gray Code Converters

Gray code converter is used for reducing the mean time between failures. The other characteristic of gray code comes when representing successive binary numbers, reflecting only one-bit change for each increment

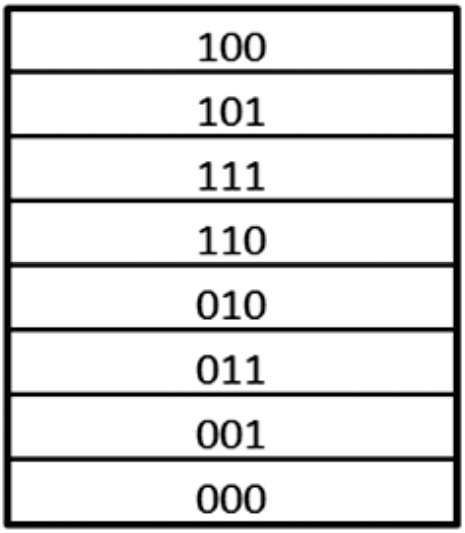

Figure 3: Gray code sequence

in binary values, thus reducing the switching activity and hence power [4]. The lower switching activity also accomplishes less glitch formation, thus reducing any metastability.

The importance of reducing the metastable condition comes when comparing the two pointers (representing 5-bit read and write addresses). By reducing the number of transitions the possibility of interpreting a signal transitioning from 1 to 0 or 0 to 1 as 1's and 0's respectively by the combinational logic (xnor gate as an equivalency check) will become easier.

3.3. Synchronizer

Overrun is critical for write operation and Underrun is critical for read operation. Full and empty conditions are critical for write and read inhibition respectively. Overrun and Underrun conditions are generated based on the read and write pointers position. Being an asynchronous design read and write operations increment respective pointers with different clock speeds. Synchronization is implemented by reading 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 depicts the synchronization mechanism. In general, a 1-bit 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. By following this mechanism read and writes clock signals are synchronized.

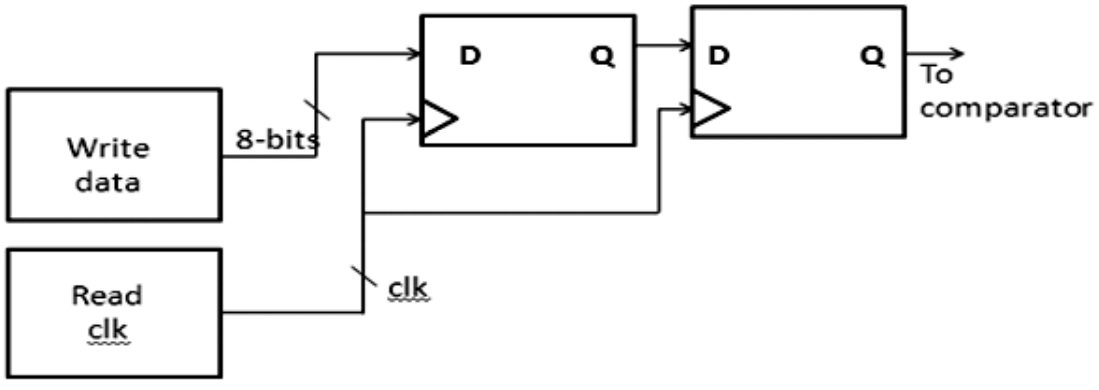

Figure 4: Synchronization mechanism

3.4. Generation of Overrun and Underrun Conditions From Asynchronous Pointers

- The Overrun and Underrun signals are intended to prevent overflow and underflow conditions i.e. requesting a write even after FIFO is full, might overwrite the data also requesting a read when FIFO is empty results in error so to prevent these errors the status flags are very crucial.
- When Overrun **(**when write pointer equals the synchronized read pointer , the last operation should be write) flag is asserted the total memory locations are filled with data and there are no free locations to write any further data, so no write operation is done. When there's **Underrun (**when read pointer equals synchronized write pointer, the last operation should be read) condition all the data is read out from the memory and the FIFO is devoid of any new data, so no read operation is done [5].

3.5. Working of Asynchronous FIFO

- 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. This being a FIFO the first loaded data will be read out first and so on.
- The memory specified in this paper can address "n" locations and 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 (Overrun condition), until read enable signal is asserted [7].
- 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 the user requests a read operation, the FIFO asserts Underrun signal and the circuit preserves the state. When there occurs read and write operations simultaneously the data that is composed into the memory first will be read out first.
- The pointers will look after the addresses, to which address of the memory the data is to be sent and from which address of the memory the data is to be read [6].

From figure 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 previous operation is write, then Overrun flag is asserted.
- If the pointers are equal and previous operation is read, then Overrun flag is asserted.

4. SIMULATION RESULTS

Simulation is being carried out on Cadence NC Simulator, using Verilog as a hardware description language. Different test cases are used to verify the Asynchronous FIFO functionality [8][9].

Figure 5 shows that Overrun flag is asserted as there's no data initially loaded into the memory and after write enable is requested there is a continuous write operation and no read operation requested until the FIFO depth is completely filled hence, Overrun flag is asserted. We can find that once the data is loaded into memory there is a de- assertion of Underrun flag. After read operation is requested we can observe that there's de-assertion of Overrun flag. Simultaneous read and write operations are also depicted in Figure 5. Hence, Asynchronous FIFO functionality is verified [10] [11][12].Figure 6 shows that Underrun flag is asserted when the FIFO is read out completely and there is no data.

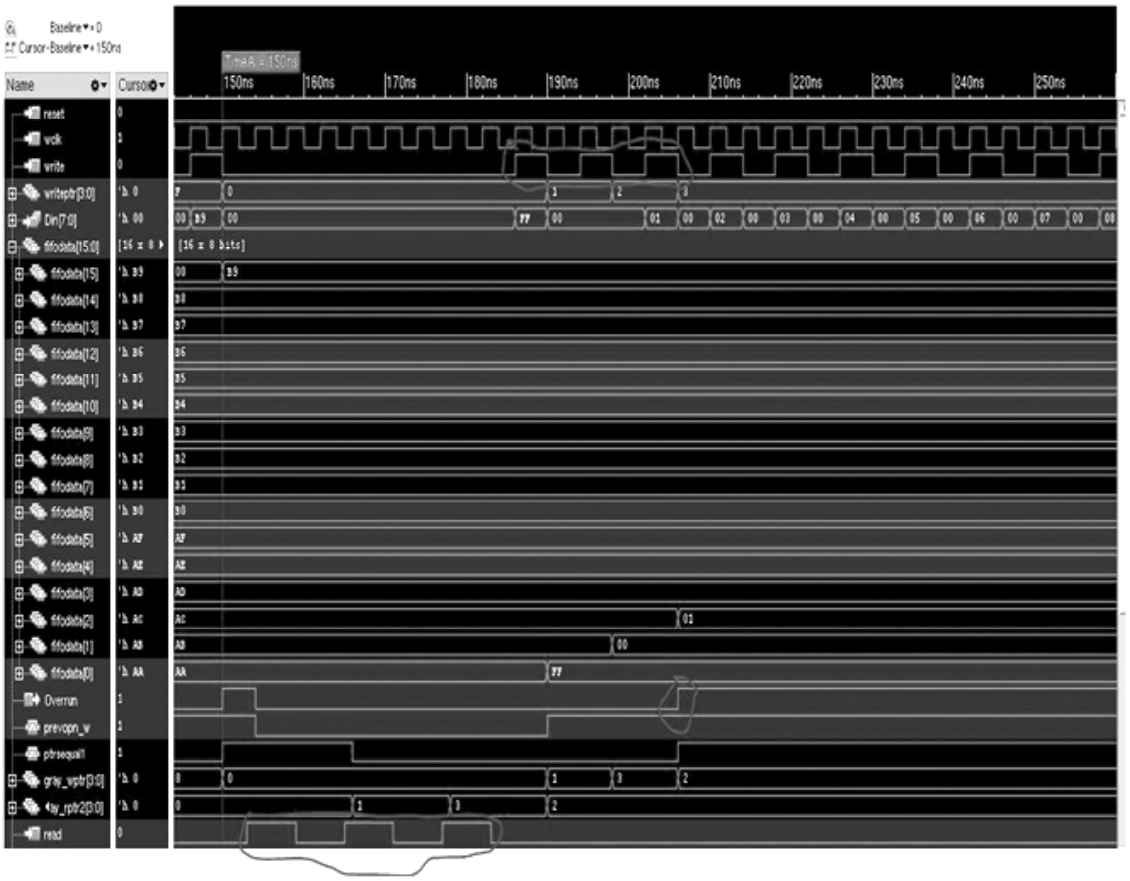

Figure 5: Overrun Flag generation

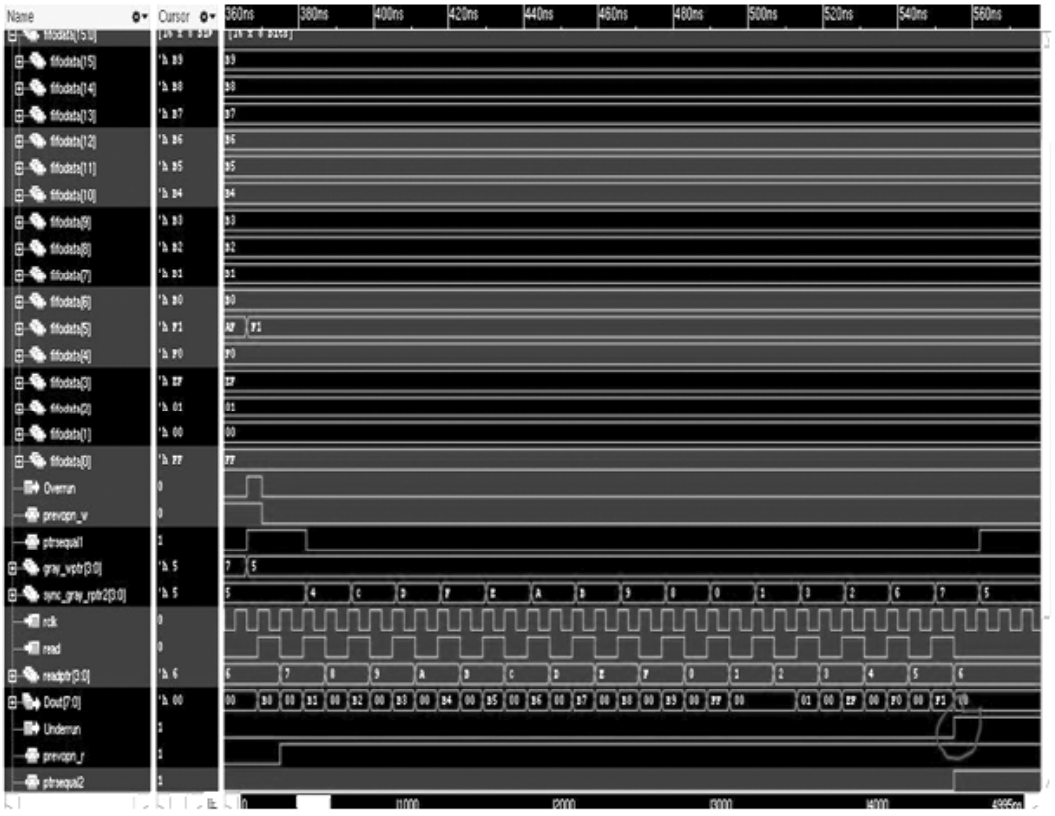

Figure 6: Underrun Flag generation

In this paper, a versatile strategy to implement a rapid Asynchronous FIFO using gray counter synchronization and with the help of previous operation is depicted. In this design the gray counters uses a comparator along with the previous operation for the generation of Overrun and Underrun status flags. This being an Asynchronous FIFO a lot of effort is required to design and meet the timing and frequencies of the read clock with the write clock and viz. This 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 Cadence NC Simulator.

5. ACKNOWLEDGEMENT

The authors would like to thank the entire semiconductor team at CYIENT and the staff of KL University for their immense support and motivation in implementing this paper. Without their guidance this paper would not have seen the light of the day. A special thanks to Mr. Ram Gollapudi (AVP, Cyient Ltd) for allowing the authors to use the companies valuable resources to complete the project.

REFERENCES

[1] http://en.wikipedia.org/wiki/FIFO_architecture.

[2] “Simulation and synthesis techniques of Async FIFO design”, available at *Sunburst design* Samir Palnitkar, “Verilog HDL: A Guide to Digital Design and Synthesis”, Second Edition.

[3] HoSuk Han, Kenneth S. Steven, *Clocked and asynchronous FIFO characterization and comparison*

[4] G. Ramesh, V. Shivraj Kumar, K. Jeevan Reddy,” *Asynchronous FIFO Design with Gray code Pointer for High Speed AMBA AHB Compliant Memory controller*”,IOSR,volume:1,issue 3, Nov-Dec 2012

[5] http://www.xilinx.com/support/documentation/ip_documentation/async_fifo.pdf [6] “Asynchronous FIFO in virtex-II FPGA’s”, available at *http://www.asicworld.com.*

[7] Asynchronous FIFO architectures by A.Nebhrajani available at,”*vlsi_book/Asynch1.pdf*”.

[8] Bergeron, Janick. Writing test benches: functional verification of HDL models. s.l.: Springer, 2003.

[9] Verilog LRM 2005 www.scribd.com/doc/6755724/Verilog-LRM.

[10] Verilog HDL Synthesis by J.Baskar.

[11] http://www.testbench.co.in.

[12] http://www.asicguru.com.