Implementation and Verification of Asynchronous FIFO using both Class-based and UVM methodologies.

Kumar Durga Manohar Karna, Mohammed Abbas Shaik,

Nivedita Boyina & Nikhitha Vadnala

Abstract

With the rapid development in the integrated circuits of the VLSI field, Asynchronous FIFO (First input First output) is primarily used to tackle with the data transmission. This paper focuses on the main problem of Asynchronous FIFO i.e; the generating FIFO full, FIFO empty, half full and half empty conditions. It prevents data loss or data corruption during the transfer process, even when the write and read clocks are completely asynchronous to each other. Dual port RAM can be read and written at any time and is very fast. The design of an Asynchronous FIFO involves several challenges, such as metastability handling, gray code pointers, and synchronization techniques. Extensive testing of asynchronous read and write operations, verifying data integrity, checking for overflow and underflow conditions, and ensuring synchronization between the write and read clocks, along with compliance with specified interface protocols. The plan includes simulations to assess both functional correctness and performance metrics, ensuring the asynchronous FIFO fits its intended role in a larger system.

Through Questa sim we have stimulated the design and verified the output, the Asynchronous FIFO can generate the data of the FIFO with FIFO full, FIFO empty, half full and half empty conditions.

1. Introduction

An Asynchronous FIFO (First in first out) is a specialized type of FIFO buffer that allows data to be read and written in different clock domains. Unlike a synchronous FIFO, where the read and write operations are controlled by the same clock signal, an Asynchronous FIFO will have separate read and write clock domains, enabling data transfer between two independently clocked systems.

The Asynchronous FIFO acts as a bridge between the sender and receiver modules, providing a reliable and efficient means of transferring data across different clock domains. It prevents data loss or data corruption during the transfer process, even when the write and read clocks are completely asynchronous to each other.

The design of an Asynchronous FIFO involves several challenges, such as metastability handling, gray code pointers, and synchronization techniques. Metastability can arise when data is transferred across asynchronous clock domains, which leads to undefined logic states. Proper metastability handling techniques, such as synchronizers, ensure reliable data transfer across clock domain boundaries. Grey cloud pointers with the property of changing only one bit at a time, which reduces the metastability. It also tracks the read and write positions in the memory. Additionally, the design incorporates synchronization techniques to properly coordinate the data transfer between the asynchronous clock domains.

2. Basic Knowledge of Asynchronous FIFO

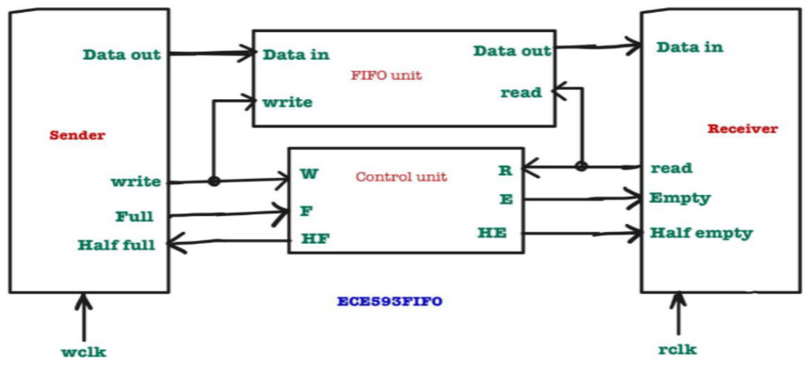
2.1 Asynchronous FIFO basic module

An asynchronous FIFO (First-In-First-Out) module is a digital logic circuit that operates as a data buffer or queue, where data is written into the module in a specific order and read out in the same order. It is called "asynchronous" because the write and read operations can occur at different clock frequencies or with no clock at all.

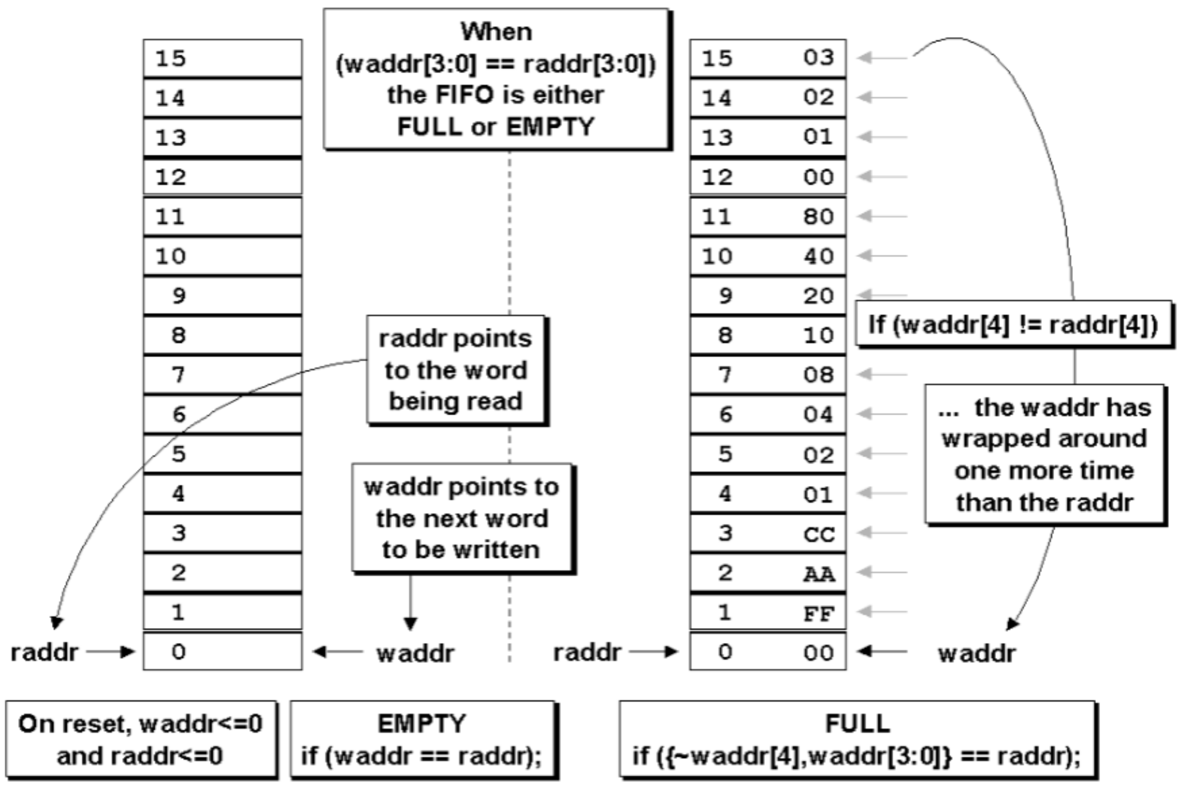
Asynchronous FIFO includes the Write, Read, Pointers, Flags and Memory.

- Write Operation: Data is written into the FIFO module when the write enable signal is asserted, and the FIFO is not full. The data is stored in memory elements, and a write pointer keeps track of the next available location to write data.
- Read Operation: Data is read from the FIFO module when the read enable signal is asserted, and the FIFO is not empty. The data is retrieved from the memory location pointed to by the read pointer, and the read pointer is updated to the next location.
- Pointers: The FIFO module has two pointers, a write pointer and a read pointer. These pointers will track the next available location for writing and reading data.
- Flags: The FIFO module has flags such as full, empty, half full and half empty to indicate the status of the FIFO.
- Memory: The FIFO module includes a memory block, typically implemented using registers or RAM, to store the data elements. The size of the memory determines the maximum number of data elements the FIFO can hold.

The depth (number of elements) and width (size of each data element) of the FIFO module can be customized based on the specific application requirements.



High Level Block Diagram of the Design System


When (waddr[3:0] == raddr[3:0]) the FIFO is either FULL or EMPTY
raddr points to the word being read
waddr points to the next word to be written
If (waddr[4] != raddr[4])
... the waddr has wrapped around one more time than the raddr
raddr
waddr
raddr
waddr
15 03
14 02
13 01
12 00
11 80
10 40
9 20
8 10
7 08
6 04
5 02
4 01
3 CC
2 AA
1 FF
0 00
On reset, waddr<=0 and raddr<=0
EMPTY if (waddr == raddr);
FULL if ({~waddr[4],waddr[3:0]} == raddr);

2.2 Key of Asynchronous FIFO design – generation of signals

In Synchronous FIFO the communication is blocked, that means the sender and the receiver must be available at the same time for communication. If the receiver is not ready to receive the data, the sender will be blocked waiting for the receiver to become available. Changes in either of the components may require changes in the other, which can increase the complexity of the system.

Asynchronous FIFO reads and writes in two different clocks, compared to synchronous FIFO reads and writes in the same clock. If the read address and write address are same, then the FIFO is empty. FIFO is full when the pointers are again equal, that is, when the write pointer has wrapped around and caught up to the read pointer. This is a problem. The FIFO is either empty or full when the pointers are equal, but which?

One design technique used to distinguish between full and empty is to add an extra bit to each pointer. Using n-bit pointers where (n-1) is the number of address bits required to access the entire FIFO memory buffer, the FIFO is empty when both pointers, including the MSBs are equal. And the FIFO is full when both pointers, except the MSBs are equal.

3. The Asynchronous FIFO Verilog Implementation

3.1. Basic Module of the Asynchronous FIFO Verilog Implementation

Read pointer design code:

```
always@(posedge rclk or negedge rrst_n)
            begin
                  if(!rst_n) begin
                        rptr <= 0;
                        wq2_rptr <= 0;
                  end
                  else begin
                        rptr <= inc_rptr;
                        wq2_rptr <= g_inc_rptr;
                  end
            end
assign inc_rptr = rptr+(read_enable & !rempty);
assign g_inc_rptr = (inc_rptr >>1)^inc_rptr;
```

Write pointer design code:

```
always@(posedge wclk or negedge wrst_n)
            begin
                  if(!wrst_n)
                  begin
                        wptr <= 0;
                        rq2_wptr <= 0;
                end
                else
                begin
                        wptr <= inc_wptr;
                        rq2_wptr <= g_inc_wptr;
                end
        end

assign inc_wptr = wptr+(write_enable & !wfull);
assign g_inc_wptr = (inc_wptr >>1)^inc_wptr;
```

Synchronizer Code:

```
always@(posedge clk)
        begin
                if(!rst_n)
                begin
                        q <= 0;
                        D_out <= 0;
                end
                else
                begin
                        q <= D_in;
                        D_out <= q;
                end
        end
```

3.2 Synchronizer

The synchronization logic uses a two-flip-flop synchronizer to transfer the gray code pointers between clock domains. On the write path, the write pointer gray code value is synchronized from the write clock domain to the read clock domain and vice versa for the read pointer.

3.3 Control Logic

The control logic manages the read and write operations ensuring that the FIFO operates correctly in various scenarios like FIFO empty, FIFO full, half full and half empty.

4. Performance

The performance of an Asynchronous FIFO involves several factors, such as metastability handling, latency. Grey cloud pointers with the property of changing only one bit at a time, which reduces the metastability when transferring data across asynchronous clock domains which leads to undefined logic states. It also tracks the read and write positions in the memory.

5. Design Testing and Verification

Verified the design by writing test cases. Verifying the functionality of the asynchronous FIFO design is crucial for ensuring its dependability and accuracy. Simulation testbenches are developed using the Verilog hardware description language to model and evaluate various scenarios the FIFO may encounter, such as different clock frequencies for the read and write domains, diverse data patterns being transferred through the FIFO, and edge cases like full or empty conditions.

6. Conclusion

In this paper, the basic model of Asynchronous FIFO is designed and implemented in SystemVerilog which effectively manages the clock domain crossings and verifying it with the UVM framework. Dual-port RAM and Gray code pointers ensure reliable and efficient data transfer between asynchronous clock domains. UVM Framework comprises of a comprehensive testbench environment to examine the behavior of the design when the data crosses from one clock domain to another. This design is vital for applications requiring robust and high-speed data communication across different clock domains.

References

1. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
2. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf
3. https://sites.google.com/view/uvmprimer-com/home
4. https://ieeexplore.ieee.org/abstract/document/8397323
5. https://repository.rit.edu/cgi/viewcontent.cgi?article=11135&context=theses