DESIGN SPECIFICATION REPORT

ECE-593: Fundamentals of Pre-Silicon Validation

Maseeh College of Engineering and Computer Science Winter 2025

Design and Verification of Asynchronous FIFO using Class Based & UVM

Github link: https://github.com/naidumaheshchowdary/Team_15_Async_FIFO

TEAM 15:

Alaina Anand Nekuri (PSU ID: 959604917)
Mahesh Naidu (PSU ID: 917048048)
Siddhartha Kaushik Gatta (PSU ID: 946785223)
Venkata Sai Dhilli (PSU ID: 980087156)

Project Name	Design and Verification of Asynchronous FIFO using Class Based & UVM
Location	Portland
Start Date	25 January 2025
Estimated Finish Date	March 11, 2025

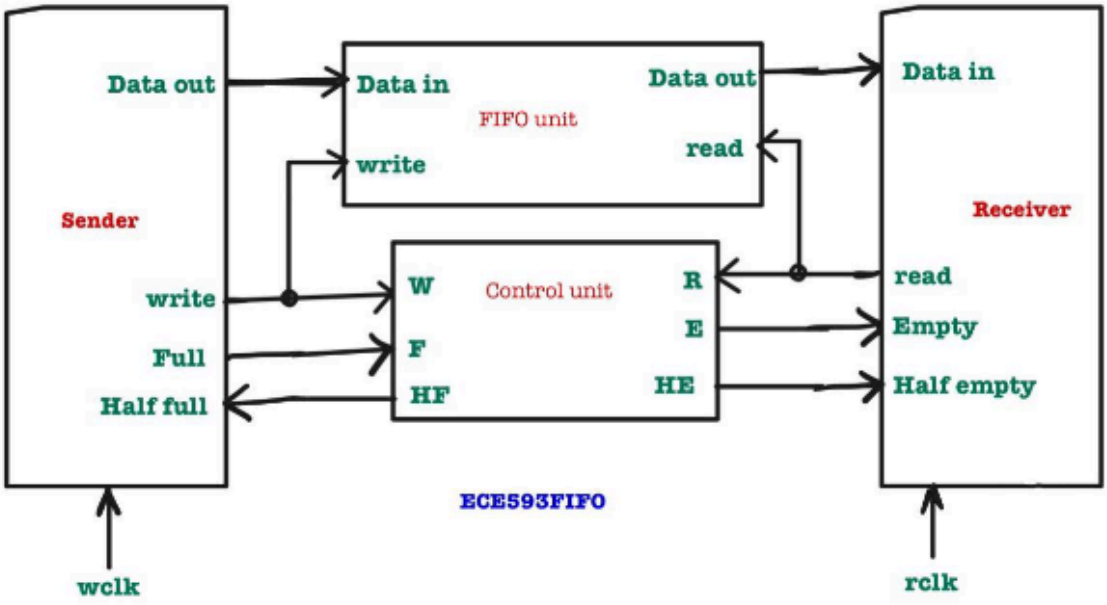
Prepared by: Team 15	
Prepared for: Prof. Venkatesh Patil	
Alaina Anand Nekuri	alainaa@pdx.edu
Mahesh Naidu	maheshn@pdx.edu
Siddhartha Kaushik Gatta	sgatta@pdx.edu
Venkata Sai Dhilli	dhilli@pdx.edu

Introduction:

In digital systems, efficient data transfer between different clock domains is crucial. An Asynchronous FIFO (First-In, First-Out) buffer is designed to handle this scenario by allowing data to be read and written using separate independent clock signals. Unlike Synchronous FIFOs, which operate under a single clock domain, Asynchronous FIFOs provide a flexible way to transfer data across systems running at different speeds without data loss or corruption.

The key advantage of an Asynchronous FIFO is that it acts as a buffer, preventing overflow and ensuring smooth communication between sender and receiver modules. However, designing such a system presents challenges such as metastability handling, synchronization, and data integrity. Metastability occurs when signals cross between asynchronous clock domains, which can lead to undefined logic states. To mitigate this, techniques such as Gray-coded pointers, synchronizers, and proper clock domain crossing methodologies should be used.

Key challenges in designing an asynchronous FIFO include Metastability Handling, pointer management, gray code encoding.



Design Specification:

The design consists of a memory buffer, a write pointer, a read pointer, and control logic to handle data movement while preventing overflow and underflow conditions. The write pointer keeps track of the memory location where the next data item will be written, while the read pointer indicates the current location from which data should be read. Since the FIFO operates in two different clock domains i.e, Writing frequency of sender is 120MHz and Read Frequency of receiver is 50MHz, it is crucial to implement a synchronization mechanism to avoid metastability issues.

During initialization or reset, both the read and write pointers are set to zero, indicating that the FIFO is empty. Data is written to the location specified by the write pointer, and after every successful write operation, the pointer is incremented. Similarly, the read pointer moves forward after every read operation. The FIFO remains empty when both pointers are equal, and the empty flag is asserted to prevent unintended reads. When the write pointer wraps around and catches up with the read pointer, the FIFO is considered full, and the full flag is asserted to prevent additional writes.

One of the key challenges in asynchronous FIFO design is distinguishing between the full and empty states, as both occur when the read and write pointers are equal. To address this, an extra bit is added to each pointer, which toggles when the FIFO wraps around. If the extra bit of the write pointer differs from that of the read pointer, it indicates that the FIFO has wrapped around and is full. On the other hand, when both pointers, including the extra bit, are equal, the FIFO is empty.

Another critical aspect of the design is the use of Gray-coded pointers for synchronization. When transferring data between different clock domains, binary counters can lead to metastability because multiple bits may change simultaneously. Using Gray code ensures that only one bit changes at a time, reducing the likelihood of metastability and improving the stability of pointer synchronization.

The design also includes logic to handle overflow and underflow conditions. When the FIFO is full, additional write operations are ignored to prevent data corruption. Likewise, when the FIFO is empty, read operations are blocked to avoid retrieving invalid data. The FIFO operates efficiently by ensuring that the read pointer always points to the next valid data, minimizing the number of clock cycles required for data retrieval.

Overall, the design of the Asynchronous FIFO focuses on efficient memory utilization, proper synchronization, and robust pointer management to provide reliable data transfer across different clock domains.

Fifo Calculation:

Writing frequency of sender = f(sender) = 120MHz.

Reading Frequency of receiver = f(receiver) = 50MHz.

Burst Length = No. of data items to be transferred = 1024.

No. of idle cycles between two successive writes is = 3.
No. of idle cycles between two successive reads is = 2.

The no. of idle cycles between two successive writes is 3 clock cycle. It means that, after writing one data,Sender is waiting for Three clock cycle, to initiate the next write. So, it can be understood that for every four clock cycles, one data is written.

The no. of idle cycles between two successive reads is 2 clock cycles. It means that, after reading one data, module B is waiting for 2 clock cycles, to initiate the next read. So, it can be understood that for every three clock cycles, one data is read.

Time required to write one data item = $4 * (1/120)$ = 33.33 nSec.
Time required to write all the data in the burst = 1024 * 33.33 nSec. = 34,129.92 nSec.
Time required to read one data item = 3*(1/50) = 60 nSec.

So, for every 60 nSec, the Receiver is going to read one data in the burst.
So, in a period of 34129.92 nSec, 1024 no. of data items can be written.

The no. of data items can be read in a period of 34129.29 nSec = 34,129.92/60 = 568.82 ≃ 569
The remaining no. of bytes to be stored in the FIFO = 1024 – 569 = 455.
So, the FIFO which has to be in this scenario must be capable of storing atleast 455 data items.
So, the minimum depth of the FIFO should be 455.

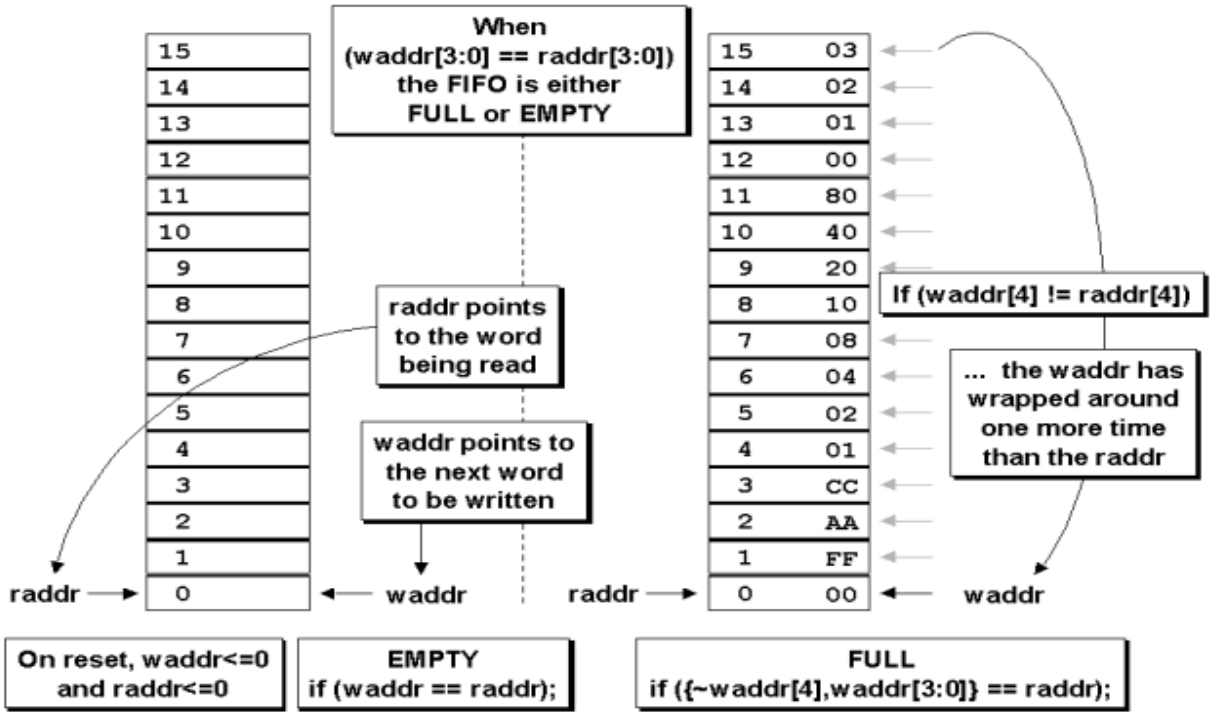


Figure 1 - FIFO full and empty conditions

References:

[1] https://hardwaregeeksblog.wordpress.com/wp-content/uploads/2016/12/fifodepthcalculationmadeeasy2.pdf

[2] http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

[3] https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/

[4] https://ieeexplore.ieee.org/document/10090696