Fundamentals of Pre-Silicon Validation || Winter -2024

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

DESIGN SPECIFICATION DOCUMENT

Project Team:
Jaswanth Pallappa, jaswanth@pdx.edu
Brunda Marpadaga, brunda@pdx.edu
Maha Lakshmi Potabattuni, mahalak@pdx.edu
Akhila Veedula, akhilav@pdx.edu

DESIGN OVERVIEW:

The asynchronous FIFO design is intricately developed to tackle the complexities inherent in asynchronous systems, particularly focusing on pointer comparisons within the FIFO structure. Through meticulous attention to detail, the design leverages simulation and synthesis techniques to achieve optimal functionality and performance.

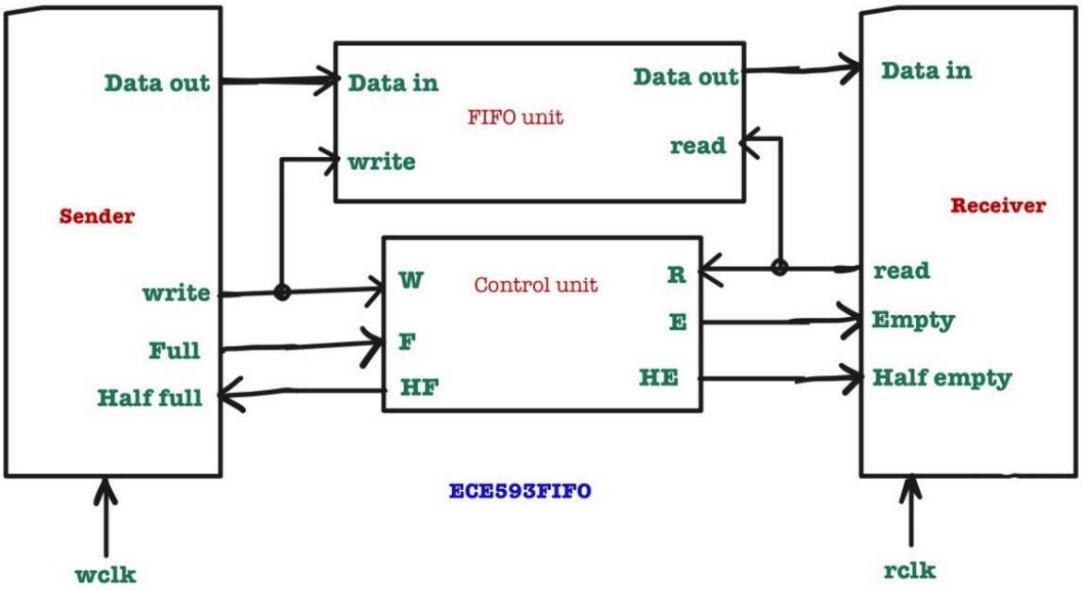

Figure 1. Block Level representation of Design

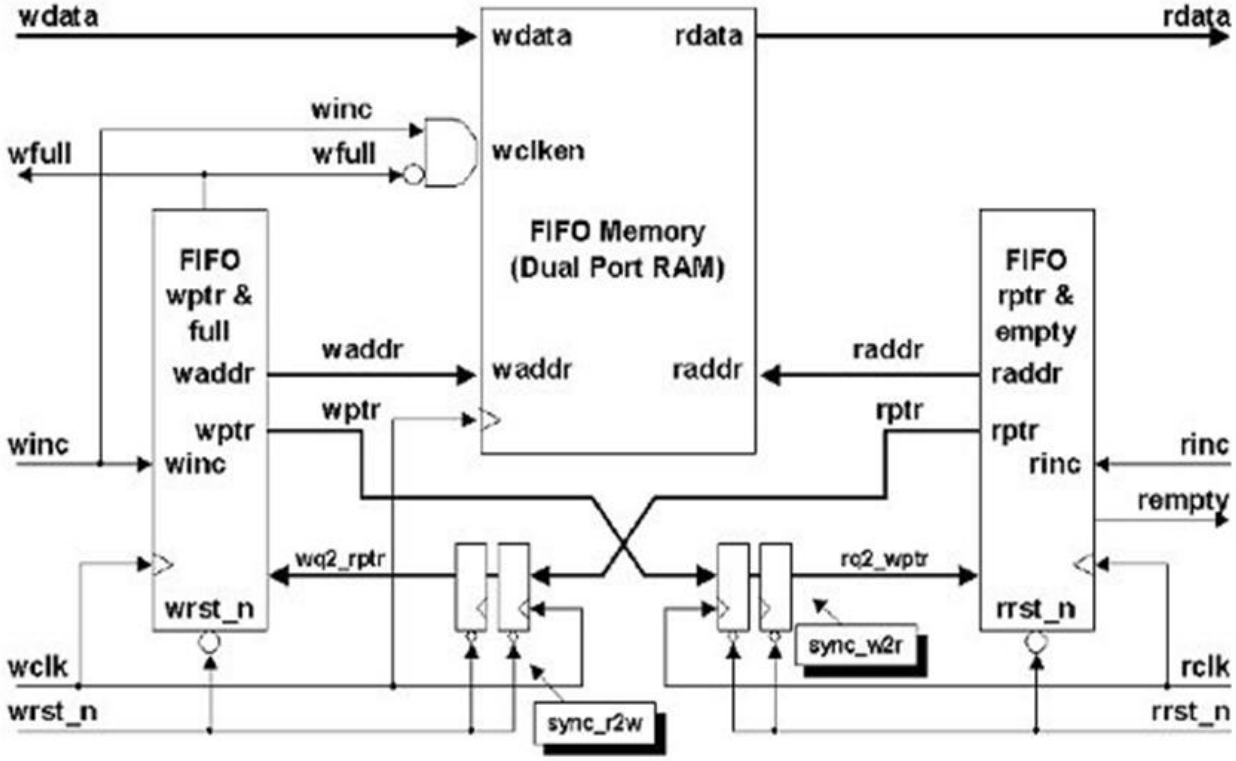

Figure 2. Architecture

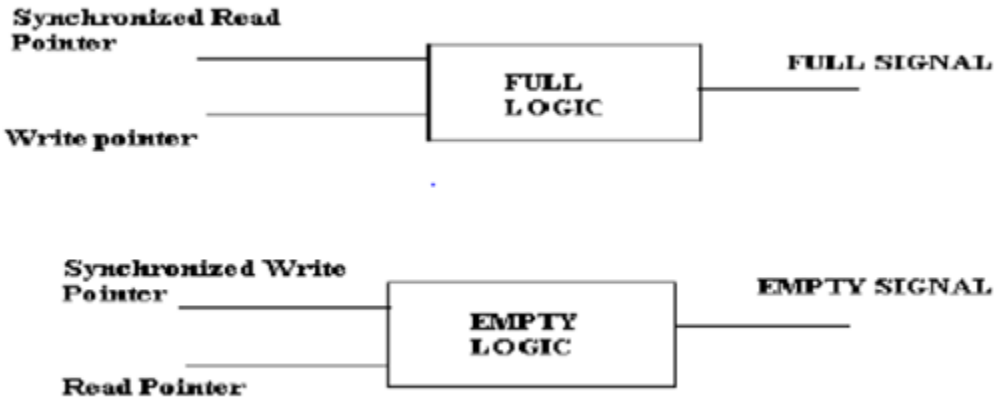

Figure 3. FULL and EMPTY logic blocks

DESIGN AND IMPLEMENTATION CONSTRAINTS:

- Sender Clock Frequency = 500 MHz = fA

Write Idle Cycles = 2 (For every consecutive clock cycle, data will be written)
Write Burst Size = 1024

- Receive Clock Frequency = 225 MHz = fB

Read Idle Cycles = 1 (For every 3 clock cycles, single data is read)

Since, fA > fB and
Given Burst Length = 1024

- The no. of idle cycles between two successive writes is 2 clock cycle. It means that, after writing one data, module A is waiting for two clock cycle, to initiate the next write. So, it can be understood that for every **three** clock cycles, one data is written.
- The no. of idle cycles between two successive reads is 1 clock cycle. It means that, after reading one data, module B is waiting for one clock cycle, to initiate the next read. So, it can be understood that for every **two** clock cycles, one data is read.
- Time required to write one data item = 3 * 1/500 MHz = 6nS
- Time required to write an entire burst of data = 1024* 6 nS. = 6,144 nS
- Time required to read one data item = 2 *1/225 MHz = 9 nS
- So, for every 9 nS, the module B is going to read one data in the burst.
- In a period of 6144 n sec, 1024 number of data items can be written.
- Number of data items can be read in a period of 6144 n sec = (6144 n sec/9)
  = 682
- Remaining number of bytes to be stored in FIFO = 1024 – 682
  = 342
- Therefore, minimum depth of FIFO = 342

IMPLEMENTATION:

FIFO consists of memory storage, write and read pointers, synchronizers, and logic blocks for determining full and empty conditions.

Below are modules and description of each module in the asynchronous FIFO design and its functionality:

1. **FIFO Memory (fifomem):**
   - Functionality: Stores the data items in a memory array with a depth of 256 bytes (adjustable). Data is written into the memory on the positive edge of the write clock (`wclk`) when the write increment signal (`winc`) is asserted and the FIFO is not full (`wfull`). Data is read from the memory on the positive edge of the read clock (`rclk`) based on the read address (`raddr`).

2. **Read Pointer Empty (rptr_empty):**
   - Functionality: Determines if the read pointer is empty. It calculates the read address (`raddr`) based on the read increment signal (`rinc`) and synchronizes it with the write pointer (`rq2_wptr`). The module also provides the read pointer value (`rptr`) and asserts the empty signal (`rempty`) when the read pointer matches the write pointer.

3. **Sync Read to Write (sync_r2w):**
   - Functionality: Synchronizes the read pointer to the write pointer clock domain. It ensures that the read pointer (`rptr`) is updated synchronously with the write pointer (`wptr`) to prevent data loss or corruption when reading data from the FIFO.

4. **Sync Write to Read (sync_w2r):**
   - Functionality: Synchronizes the write pointer to the read pointer clock domain. Similar to `sync_r2w`, this module ensures that the write pointer (`wptr`) is synchronized with the read pointer (`rptr`) clock domain, maintaining proper operation of the FIFO in asynchronous systems.

5. **Write Pointer Full (wptr_full):**
   - Functionality: Determines if the write pointer is full. It calculates the write address (`waddr`) based on the write increment signal (`winc`) and synchronizes it with the read pointer (`wq2_rptr`). The module also provides the write pointer value (`wptr`) and asserts the full signal (`wfull`) when the write pointer matches the read pointer.

6. **Top Module (async_fifo1):**
   - Functionality: Integrates all the above modules to create the complete asynchronous FIFO design. It handles the interactions between the modules and provides input and output ports for interfacing with the external environment.

Each module plays a crucial role in ensuring the correct operation of the asynchronous FIFO design, facilitating efficient data storage and retrieval in asynchronous systems.