Fundamentals of Pre-Silicon Validation Winter-2025

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

DESIGN SPECIFICATION DOCUMENT

Team -1
Bhargav Chunduri - bhargavc@pdx.edu
Dhushyanth Dharmavarapu – dharmava@pdx.edu
Venkata Krishna Kumar vedantam – vedantam@pdx.edu

DESIGN OVERVIEW:

To address the intricacies of asynchronous systems, the asynchronous FIFO design is carefully crafted, with a special emphasis on pointer comparisons inside the FIFO structure. The design uses synthesis and simulation methods to obtain the best possible functionality and performance through painstaking attention to detail.

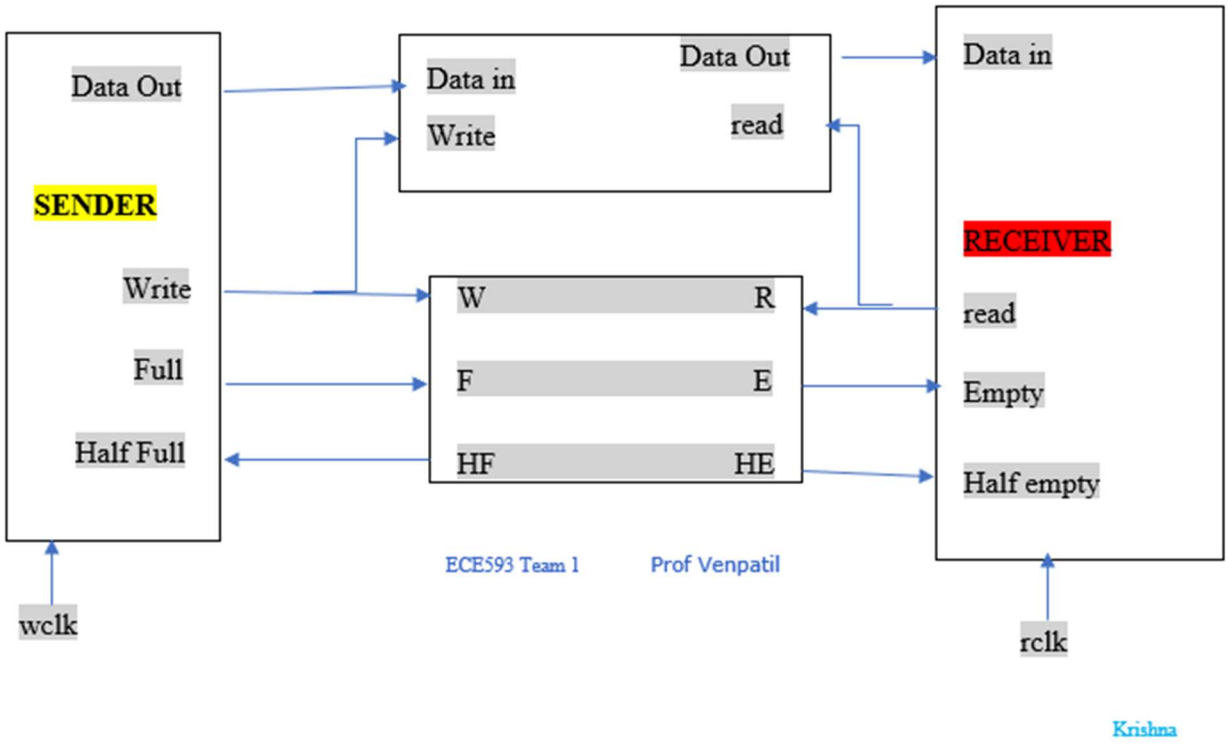

Figure 1. Block Level representation of the Design

DESIGN AND IMPLEMENTATION CONSTRAINTS:

- We have two modules operating at different clock frequencies:
- Sender (Module A): 500 MHz (fA)
- Receiver (Module B): 225 MHz (fB) Write and Read Conditions:

• Writing: After every write operation, Module A waits 2 clock cycles before writing the next data. This means data is written once every 3 clock cycles.

• Reading: After each read operation, Module B waits 1 clock cycle before reading the next data. This means data is read once every 2 clock cycles. Since fA > fB and with a burst length of 1024:

Timing Calculations:
• Time to write one data item: 3×1500 MHz=6 ns
• Time to write an entire burst (1024 data items): 1024×6 ns=6144
• Time to read one data item: 2×1225 MHz=9
• Total data read in 6144 ns: 6144÷9=682 data items
• Data remaining in FIFO: 1024−682=3421024 - 682 = 342

FIFO Depth Requirement:
Since the receiver is slower than the sender, some data will accumulate in the FIFO. To ensure smooth operation, the minimum FIFO depth should be at least 342 entries to store the extra data. So for easiness of calculation of we taken depth as 512 which is 2 power 9.

IMPLEMENTATION:

The FIFO design incorporates memory storage, write and read pointers, synchronizers, and logic blocks for managing full and empty conditions. Below are the modules and their functionalities in the asynchronous FIFO design:

1. **FIFO Memory (fifomem)**:
    - **Functionality**: Stores data items in a memory array with adjustable depth (256 bytes). Data is written into the memory on the rising 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 rising 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 computes the read address (`raddr`) using 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 with the write pointer clock domain. It ensures that the read pointer (`rptr`) is updated synchronously with the write pointer (`wptr`) to avoid data loss or corruption when reading from the FIFO.
4. **Sync Write to Read (sync_w2r)**:
    - **Functionality**: Synchronizes the write pointer with the read pointer clock domain. This module ensures that the write pointer (`wptr`) is aligned with the read pointer (`rptr`), facilitating proper FIFO operation in asynchronous systems.
5. **Write Pointer Full (wptr_full)**:

- **Functionality**: Determines if the write pointer is full. It calculates the write address (`waddr`) using 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 aforementioned modules to form the complete asynchronous FIFO design. It manages the interaction between the modules and provides input and output ports for interfacing with the external environment.

For the asynchronous FIFO architecture to function properly and guarantee effective data storage and retrieval in asynchronous systems, each module is necessary.

.