Fundamentals of Pre-Silicon Validation - Winter -2024

Implementation and Verification of Asynchronous FIFO

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

DESIGN SPECIFICATION DOCUMENT

DESIGN OVERVIEW:

The asynchronous FIFO design is carefully crafted to address the challenges of asynchronous systems, with a particular emphasis on pointer comparisons within the FIFO structure. By employing precise simulation and synthesis techniques, the design ensures optimal functionality and performance.

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**.

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:

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):**
   - The FIFO stores data items in a memory array with a configurable depth of **256 bytes**. Data is written into the memory on the **rising edge of the write clock (w_clk)** when both the **write increment signal (win_c)** is asserted and the FIFO is **not full (w_full)**. Similarly, data is read from memory on the **rising edge of the read clock (r_clk)** based on the specified **read address (raddr)**.

2. **Read Pointer Empty (readptr_empty):**
   - Checks whether the **read pointer** is empty by computing the **read address (raddr)** based on the **read increment signal (r_inc)** and synchronizing it with the **write pointer (rq2_wptr)**. The module outputs the **read pointer value (r_ptr)** and asserts the **empty signal (rempty)** when the read pointer matches the write pointer.

3. **Sync Read to Write (sync_read2w):**
   - Synchronizes the read pointer to the write pointer clock domain. It ensures that the read pointer (`rptr`) is updated synchronously with the write pointer (`w_ptr`) to prevent data loss or corruption when reading data from the FIFO.

4. **Sync Write to Read (sync_write2r):**
   - Synchronizes the write pointer to the read pointer clock domain. Similar to `sync_read2w`, this module ensures that the write pointer (`wptr`) is synchronized with the read pointer (`r_ptr`) clock domain, maintaining proper operation of the FIFO in asynchronous systems.

5. **Write Pointer Full (writeptr_full):**
   - Checks whether the **write pointer** is full by computing the **write address (waddr)** based on the **write increment signal (winc)** and synchronizing it with the **read pointer (wq2_rptr)**. The module outputs the **write pointer value (wptr)** and asserts the **full signal (wfull)** when the write pointer matches the read pointer.

6. **Top Module (async_fifo):**
   - Combines all the aforementioned modules to form the complete **asynchronous FIFO design**. It manages the interactions between the modules and provides input and output ports for communication 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.