ECE-593

Fundamentals of Pre-Silicon Validation

Asynchronous FIFO

TEAM-14

Team Members:

Sai Kumar Pulagam Email id: spulagam@pdx.edu PSU ID: 984110186	**Ganeshkumar Sanke** Email id: ganeshs@pdx.edu PSU ID: 947941776
Sandeep Reddy Lingala Email id: lingala@pdx.edu PSU ID: 988094614	**Reshwanth Sri Sai Sesham** Email id: sesham@pdx.edu PSU ID: 964563316

Design specifications with calculations:

Sender Clock Frequency = 250MHz

- Number of idle cycles between two successive writes = 1
- Receiver Clock Frequency = 100MHz
- Number of idle cycles between two successive reads = 3
- Write Burst = 200
- Sender Clock Frequency > Receiver Clock Frequency.

The number of idle cycles between two successive writes is 1 clock cycle, which means after writing one data, write module is waiting for 1 clock cycles to initiate the next write, meaning every 2 clock cycles one data is written.

- The number of idle cycles between two successive reads is 3 clock cycles, which means after reading one data, read module is waiting for 4 clock cycles to initiate next read, meaning every 4 clock cycles on data is read. Time required to write one data item = 2 * (1/250MHz) = 8 ns.

Time required to write the data in the burst = 1600ns.

Time required to read one data item = 4 * (1/100MHz) = 40ns - For every 40ns read module is going to read one data item in the burst.

No of data items can be read in a period of 1600ns = (1600/40) = 40 items. - Remaining no of bytes to be stored in the FIFO = 200 – 40 = 160

The minimum depth of asynchronous FIFO is 160 bytes. It has been given adjusted to the next nearest powers of 2. So now, the depth of our FIFO is 256 (2 to the power 8).

The width of asynchronous FIFO is assumed to be 8 bits.

All the above-mentioned values can be parameterized for varying size.

The FIFO will be using active low reset in both write and read clock domains.

Data will be written at positive edge of wclk and data will be read at the positive edge of rclk.

FIFO = 256 Elements Full condition. FIFO = 256 Elements Full condition.

Implementation of design:

Overview:

The project aims to develop an Asynchronous FIFO memory structure, which is crucial for enabling data transfer between different clock domains in our system. This FIFO design is divided into several key components, each responsible for a specific function related to the FIFO's operation.

Design Components:

Fifo_mem: The fifo_mem module is responsible for managing FIFO data transactions with adjustable parameters such as data width (DATASIZE) and memory depth (ADDRSIZE). It allows write operations when the memory is not full and supports direct memory reads. This design ensures smooth data flow and preserves integrity across different clock domains.

Rptr_empty: The rptr_empty module oversees the read pointer and monitors the empty status of the FIFO. It dynamically updates the read pointer in response to read requests and employs Gray code to ensure robust communication across diverse clock domains.

Sync_r2w: This module guarantees the stable transfer of the read pointer to the write domain using a dual-register synchronization technique, minimizing metastability risks between clock domains.

Sync_w2r: The sync_w2r module synchronizes the write pointer to the read domain using a dual-register configuration, ensuring updated and stable pointer values for dependable read operations.

Wptr_full: The wptr_full module manages the write pointer and indicates FIFO full status. It oversees safe data writes and pointer updates through binary to Gray code conversion.

The async_fifo module in top.sv orchestrates essential submodules to create a comprehensive asynchronous FIFO system with customizable data size (DSIZE) and address size (ASIZE). It coordinates interactions among the FIFO memory (fifomem), pointer controls (sync_r2w and sync_w2r), and status modules (wptr_full and rptr_empty), facilitating both write and read operations. This ensures synchronized data transfer across distinct clock domains while maintaining integrity and flow control.

The tb.sv testbench for async_fifo simulates the FIFO by injecting signals such as winc and rinc and toggling clocks. It operates the FIFO, verifies inputs and outputs, and conducts tests to ensure reliability and correctness across various scenarios.

Instructions on how to run the simulation:

To initiate the simulation, navigate to the directory containing the run.do file. Then, run the command do run.do in Questa Sim to execute the script, which encompasses all the necessary instructions for the simulation. Questa Sim will carry out these commands, and you can view the results within its interface for additional analysis.

Verification plan:

Objective:

To verify the FIFO's capability to handle data transfer across asynchronous clock domains while preserving data integrity and synchronization.

Environment Setup:

1. Utilization of SystemVerilog and UVM (Generator, Driver, Monitor, Scoreboard) for the testbench framework to construct a verification environment.
2. Incorporation of the async_fifo top module, comprising fifo_mem, rptr_empty, sync_r2w, sync_w2r, and wptr_full submodules.

Test Cases:

Initialization and Reset Test: Verification of proper initialization of all FIFO signals and functionality of the reset feature across all modules.

Write Operation Test: Confirmation of the FIFO's correct acceptance and storage of data when the write enable signal is active, ensuring accurate data alignment and storage without loss.

Read Operation Test: Validation of the FIFO's ability to output valid data upon activation of the read enable signal, with data matching the expected output.

FIFO Full Condition Test: Testing the FIFO's capability to recognize and handle full conditions accurately by attempting to write beyond its capacity.

FIFO Empty Condition Test: Assessment of the FIFO's empty detection by continuously reading data until it indicates an empty condition.

Coverage Metrics:

1. Monitoring of code coverage to guarantee the execution of all lines of code within the submodules.
2. Tracking of functional coverage to ensure comprehensive testing of all specified functionalities, including edge cases for write and read operations (Coverage Monitor).
3. Implementation of assertion-based checks to automatically verify correct behavior during simulations.

Regression Testing:

Development of a set of regression tests to be executed after every significant codebase change, ensuring the preservation of existing functionality against updates.

Git Repository Link: **https://github.com/rsesham18/Asynchronous-FIFO**

References:

- ***Venkatesh Patil-"ECE-593-Lecture Slides".***
- ***https://verificationguide.com/uvm/uvm-testbench-architecture/***
- ***https://www.chipverify.com/***
- ***https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/***
- ***https://vlsiuniverse.blogspot.com/2013/09/synchronization-schemes.html***
- ***https://www.verilogpro.com/asynchronous-fifo-design***
- ***Clifford E. Cummings, Sunburst Design, "Simulation and Synthesis Techniques for Asynchronous FIFO Design" http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf***
- ***https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf***