ECE593, Spring 2025

Verification Plan

DUT Specifications:

Data can be safely passed from one asynchronous clock domain to another using FIFO. When two clock domains are asynchronous to one another, an asynchronous FIFO design occurs when data values are written from one clock domain to a FIFO buffer and read from the same FIFO buffer from another clock domain.

The next word to be written is always shown by the write pointer. The write pointer increases and the empty flag is cleared as soon as the first piece of data is written to the FIFO. The current FIFO word to be read is always shown by the read pointer.

When the read and write pointers are equal, the FIFO is empty.

FIFO Depth Calculation:

Double the frequency of write and half of the read

120 MHz is the sender clock frequency.

Three idle cycles separate two consecutive writes, and the receiver clock frequency is 50 MHz.

There are two idle cycles in between two consecutive reads.

Given that the sender clock frequency is less than the receiver clock frequency, write burst = 1024.

Three clock cycles separate two consecutive writes, indicating that after writing one piece of data, the write module waits three clock cycles before starting the next write. This means that one piece of data is written every four clock cycles.

Writing one data item takes 4 * (1/120MHz) = 33.33 ns.

It took 34,129.92 = 34,129 ns to write the data in the burst.

One data item's reading time is equal to 3 * (1/50MHz) = 60 ns.

Thus, the read module will read one data item from the burst every 60 ns.

In 34,129 ns, the number of data pieces that can be read is 34,129/60 = 568.81 = 568

items. 1024 – 568 = 456 is the remaining number of bytes that must be kept in the FIFO.

GIT LINK :https://github.com/xbicai/ECE593_Project

Roles & Responsibilities:

Role	Member
High Level Design Specification (HLDS)	Pavan
Design spec calculation and plan	Suryateja
FIFO Memory module	Gene Hu
Read pointer Empty module	Gene Hu, Suryateja
Basic Testbench	Kai Roy
Write pointer Full module	Kai Roy, Pavan

- **Strategy**: Dynamic Simulation
    - **Why Chosen**: Suitable for verifying asynchronous behavior, flag updates, and handshake signals. Formal verification may be added for pointer synchronization.
- **Driving Methodology**: Combination of directed and constrained-random tests.
- **Test Generation Methods**:
    - **Directed Tests**: Reset, basic write/read, fill/empty scenarios.
    - **Constrained Random**: Random DIN, random delays between read/write operations.
- **Checking Methodology**:
    - From specification: Verify flags, handshake signals, and count vectors.
    - Automated checkers compare actual vs. expected outputs.

Testbench Architecture

- **Components**:
    - **Driver**: Generates DIN, WR_EN, RD_EN, AINIT synchronized to WR_CLK and

RD_CLK. Uses random DIN values.
    - **Monitor**: Observes DOUT, FULL, EMPTY, ALMOST_FULL, ALMOST_EMPTY, WR_ACK, WR_ERR, RD_ACK, RD_ERR, WR_COUNT, RD_COUNT.
    - **Scoreboard**: Tracks written data and compares with read data to verify integrity. Maintains expected FIFO state (e.g., occupancy).
    - **Checker**: Automates comparison of actual vs. expected outputs (replacing manual $display statements).
- **Functions**:
    - **Reset Functionality**: AINIT resets FIFO to empty state (EMPTY = 1, FULL = 0, ALMOST_FULL = 0, ALMOST_EMPTY = 1, WR_COUNT = 0, RD_COUNT = 0).
        - ALMOST_EMPTY: Asserted when one read remains.
    - **Count Vectors**: WR_COUNT and RD_COUNT accurately reflect FIFO occupancy.
    - **Asynchronous Operation**: Correct data transfer and flag updates across WR_CLK and RD_CLK.
    - **Invalid Request Rejection**: Reject writes when FULL = 1 and reads when EMPTY = 1 without affecting FIFO state.
- **Unverified Functions**:
    - None. All specified functions (flags, handshake signals, count vectors) will be verified to ensure complete functionality.
- **Critical Functions for Tapeout**:
    - Reset, Write, Read, Flag Updates (FULL, EMPTY), Handshake Signals (WR_ACK, WR_ERR, RD_ACK, RD_ERR).
- **Non-Critical Functions**:
    - ALMOST_FULL, ALMOST_EMPTY, WR_COUNT, RD_COUNT (useful but not essential for basic operation).

Required Tools

- **Software Tools**:
    - **Simulator**: QuestaSim or VCS for SystemVerilog simulation.
    - **Coverage Tools**: QuestaSim/VCS coverage analysis for code and functional coverage.

Mitigation Plans:

- Use constrained-random tests to stress clock domain crossings.
- Implement comprehensive coverage metrics and review reports weekly.
- Schedule tool access and optimize testbench for simulation efficiency.

Risks:

    - **Asynchronous Clock Synchronization**: Metastability in pointer synchronization

across clock domains.
    - **Coverage Gaps**: Missing corner cases (e.g., simultaneous read/write with maximum dept

Functions to be Verified

Functions from Specification and Implementation

- **Verified Functions**:
    1. **Reset Functionality**: AINIT resets FIFO to empty state (EMPTY = 1, FULL = 0, ALMOST_FULL = 0, ALMOST_EMPTY = 1, WR_COUNT = 0, RD_COUNT = 0).
    2. **Write Operation**: Write data to FIFO when WR_EN = 1 and FULL = 0. Assert WR_ACK on success, WR_ERR on failure.
    3. **Read Operation**: Read data from FIFO when RD_EN = 1 and EMPTY = 0. Assert RD_ACK on success, RD_ERR on failure.
    4. **Flag Updates**:
        - FULL: Asserted when FIFO has DEPTH entries.
        - EMPTY: Asserted when FIFO has 0 entries.

References / Citations / Acknowledgements

- **References**:
    - Xilinx Asynchronous FIFO V3.0 Design Specification.
    - SystemVerilog IEEE 1800-2017 Standard.
    - ECE-593 Course Materials, Winter 2025.