Asynchronous FIFO: Verification Plan

Session 1 Group 4

Nick Allmeyer, Alexander Maso, Ahliah Nordstrom

Version 1.1 - April 22, 2024

Table of Contents

0: Document Revision History

Version	Date	Description
v 1.1	4/22/2024	For Milestone 1 deliverable

1: Function Intent, Design Description, and Specification

The asynchronous FIFO acts as a buffer to manage data flow between two parts of a system operating on different clock domains. The FIFO ensures data integrity and timing without the need for clock synchronization between sender and receiver blocks, operating at 80 Mhz and 50 Mhz respectively.

A memory buffer with a depth of at least 45 items is used to store data elements. Operations are consistent and predictable for both READ and WRITE, as they are fully synchronous to their respective clocks. Data is written to the FIFO at a rate of 12.5 ns per operation and is determined by the rising edge of the WRITE clock. Data is read at a rate of 20 ns per operation and is determined by the rising edge of the READ clock. The ratio of WRITEs to READs is 8:5, simplified down from 120:75. This ratio defines the operational balance between input and output operations so that the FIFO does not overflow/underflow under normal operating conditions.

A binary counter is used to track READ and WRITE pointers that then indicate the next positions for READ and WRITE operations. These pointers will be initialized to zero upon a system reset and then increment as data is written and read to and from the FIFO. If an invalid READ operation is performed, due to an empty FIFO, the RD_ERR signal will be asserted. If an invalid WRITE operation is performed, due to a full FIFO, the WR_ERR signal will be asserted. The HALF_EMPTY and HALF_FULL flags are used as a way to indicate nearing their respective states. The FULL flag is generated in the write-clock domain when no more data can be written to the FIFO without overwriting existing data. The EMPTY flag is generated in the read-clock domain when there are no data items to read from the FIFO and both pointers are equal.

2: Verification Levels

There will be two verification levels for the DUT: Unit-level and System-level.

- Unit-Level Verification: Each component of the DUT will be verified independently
    - Sender logic (WRITE control)
    - Receiver logic (READ control)
    - Synchronization logic for WRITE to READ
    - Synchronization logic for READ to WRITE
    - FIFO memory logic
- System-Level Verification: All components that make up the DUT are verified together, cohesively. It is to test the interaction rather than particular functions within the DUT.

3: Required Tools

- Questasim simulator for compiling, debugging, simulating, and coverage analysis
- Choice of HDL language is SystemVerilog (SV)
- Assertion-based verification will be in SystemVerilog Assertions (SVA)
- Github and Google Drive for version control and collaboration

4: Risks, Dependencies and Mitigation Plan

Here are the expected risks and dependencies throughout the verification process:

- Underestimating amount of time to debug DUT
    - Mitigation Plan: Allocate more than expected time in schedule
- Having a hard time connecting to remote lab and/or facing delays, freezes
    - Mitigation Plan: Start deliverables early and avoid peak hours
- Making design changes late into the verification process (architecture closure)
    - Mitigation Plan: Use version control tools to update verification plan early rather than later. Begin debugging DUT as early as possible

5: Functions to be Verified

5.1 Critical Functions

These functions are to be verified before all others, as these provide the base set of tasks and behaviors of the DUT.

- Signaling of FULL and EMPTY states
- Handling of WR ratios
- FIFO Depth and overflow/underflow
- Data integrity during transfers
- Reset Behavior
- Pointer Synchronization
- Idle cycle management

5.2 Secondary Functions

These functions are not critical to the next level of verification. If one of these functions are broken, the design is not “dead”.

- Operation robustness like invalid and error conditions

5.3 Non-Verified Functions

There are no functions in the DUT that may not be applicable at any level of verification or they have already been verified.

6: Tests and Methods

6.1 Type of Verification

Gray box verification will provide the ideal level of observation and controllability of the DUT. Observation points will be used to track any interesting behaviors during simulation.

6.2 Verification Strategy

Constrained random testing to generate test inputs that are constrained by the FIFO specifications but random with those constraints for the breadth of testing.

6.3 Abstraction Level

The abstraction level that will be used is Register Transfer Level (RTL) based, since we want to ensure the FIFO functions correctly post-synthesis. Creating the FIFO from scratch might be the main driving point for choosing this abstraction level, and with consideration of our three-man team and limited timeframe for completion.

Transaction level abstraction should also be expected to be used in order to concentrate on the data bursts and R/W operations and ensure proper synchronization.

6.4 Coverage Requirements

The intended stimulus goals are as follows:

- Including metrics for functional coverage, code coverage, and assertion coverage to get a sense of our verification quality and minimize coverage gaps
- Checking all critical data paths that affect data flow from WRITE to READ
- Checking that FIFO meets required throughput rates under various test conditions

6.5 Test Scenarios: Matrix

Test Reference #	Test Description	Function	Cross-Reference
T001	Validate signaling of FULL and EMPTY states	Signaling of FULL and EMPTY states	Functional Req, Coverage Goal for state signaling
T002	Test FIFO at various depth levels	FIFO Depth, overflow/underflow	Functional Req, Coverage Goal for depth handling
T003	Validate data integrity across transfers	Data integrity during transfers	Functional Req, Coverage Goal for data integrity
T004	Observe FIFO behavior following reset operations	Reset Behavior	Functional Req, Coverage Goal for reset behavior
T005	Test synchronization of READ and WRITE pointers	Pointer Synchronization	Functional Req, Coverage Goal for pointer sync
T006	Monitor FIFO during idle cycles	Idle cycle management	Functional Req, Coverage Goal for idle management
T007	Handling different WRITE/READ operation ratios	Handling of W/R ratios	Functional Req, Coverage Goal for W/R ratio handling
T008	Operation under invalid and error conditions	Operation robustness	Functional Req, Coverage Goal or error condition handling
T009	Boundary condition testing one element from full/empty	Critical boundary condition testing	Functional Req, Coverage Goal or boundary conditions
T010	High-frequency write/read stress test	FIFO performance under stress	Functional Req, Coverage Goal for performance testing
T011	Random reset during operation	Robustness to resets	Functional Req, Coverage Goal for random reset handling
T012	Abrupt changes in READ/WRITE rates	Pointer synchronization under stress	Functional Req, Coverage Goal for dynamic sync testing
T013	Throughput verification under varied conditions	Throughput and performance metrics	Functional Req, Coverage Goal for throughput testing

Table 1: Test Case Scenarios Matrix outlining test description and cross-reference to function and coverage list

6.6 Resources

Our team is made up of three graduate students. Work will be divided equally and tasks will be assigned early in order to coordinate best with due dates. Each student will need a computer in order to run simulations and track progress through our decided tools. One of these tools,

QuestaSim, requires a license that is offered to students through PSU. By logging into the remote lab, access will be given.

7: Schedule and Responsibilities

7.1 Week of 4/15-4/21/2024

This week's focus is on getting together as a team and going through project documents in order to determine what our final project will be. This is important as Milestone #1 will be due. Here are the team responsibilities for the week:

Nick Allmeyer:

- Read Clifford E. Cummings FIFO1 and FIFO2 papers
- Meet with team on 4/17 to discuss FIFO choice
- Meet with team on 4/19 to discuss FIFO implementation

Alexander Maso:

- Read Clifford E. Cummings FIFO1 and FIFO2 papers
- Meet with team on 4/17 to discuss FIFO choice
- Meet with team on 4/19 to discuss FIFO implementation

Ahliah Nordstrom:

- Read Clifford E. Cummings FIFO1 and FIFO2 papers
- Meet with team on 4/17 to discuss FIFO choice
- Meet with team on 4/19 to discuss FIFO implementation

7.2 Week of 4/22-4/28/2024

This week's focus is on completing Milestone #1 and starting Milestone #2 deliverables for submission on 4/24 and 4/30, respectively. Here are the team responsibilities for the week:

Nick Allmeyer:

- Perform FIFO calculations with our teams specifications
- Create conventional testbench for DUT

Alexander Maso:

- Create initial DUT files
- Ensure DUT files compile and debug if necessary

Ahliah Nordstrom:

- Create Github repository and Google Drive for team

- Put together and Verification Plan and Specifications documents
- Update documents later in week if there are any nuances
- Support others when needed

7.3 Week of 4/29-5/5/2024

This week's focus is on completing Milestone #2 deliverables for submission on 4/30. Here are the team responsibilities for the week:

Nick Allmeyer:

- Test FIFO top module within class based testbench
- Test FIFO read/write pointers within class based testbench
- Support others when needed

Alexander Maso:

- Test FIFO read/write pointers within class based testbench
- Polish the Class Based testbench and ensure all interfaces are tested
- Support others when needed

Ahliah Nordstrom:

- Update version control sites and Verification Plan
- Test control and memory unit interface within class based testbench
- Support others when needed

7.4 Week of 5/6-5/12/2024

This week's focus is on completing Milestone #3 deliverables for submission on 5/14. Here are the team responsibilities for the week:

Nick Allmeyer:

- Create a working driver and monitor for complete class based verification testing
- Polish off complete class-based testbench and ensure all components are defined and working
- Support others when needed

Alexander Maso:

- Create a working transaction tester for complete class based verification testing
- Finalize any RTL changes, ensure coverage goal is as close to 100% as possible
- Support others when needed

Ahliah Nordstrom:

- Update version control sites and Verification Plan
- Create a working scoreboard for complete class based verification testing
- Support others when needed, particularly with ensuring all components are defined and working in testbench

7.5 Week of 5/13-5/19/2024

This week's focus is on completing Milestone #3 and starting Milestone #4 deliverables for submission on 5/14 and 5/28 respectively. Here are the team responsibilities for the week:

Nick Allmeyer:

- Create a working driver and monitor for complete class based verification testing
- Polish off complete class-based testbench and ensure all components are defined and working
- Support others when needed

Alexander Maso:

- Create a working transaction tester for complete class based verification testing
- Finalize any RTL changes, ensure coverage goal is as close to 100% as possible
- Support others when needed

Ahliah Nordstrom:

- Update version control sites and Verification Plan in light of any nuances
- Finish working scoreboard for complete class based verification testing
- Support others when needed, particularly with ensuring all components are defined and working in testbench

7.6 Week of 5/20-5/26/2024

This week's focus is on completing Milestone #4 deliverables for submission on 5/28. Here are the team responsibilities for the week:

Nick Allmeyer:

- Ensure that UMV_MESSAGING and UMV_LOGGING mechanisms are utilized
- Collect log and reports data from UVM mechanisms
- Ensure testbench respects UVM testbench architecture and create monitor in testbench

Alexander Maso:

- Create skeleton for UVM testbench architecture for the FIFO
- Incorporate UVM sequencer and driver to testbench
- Ensure testbench respects UVM testbench architecture

Ahliah Nordstrom:

- Update version control sites
- Incorporate UVM scoreboard and FIFO interfaces in the testbench
- Add UVM section in Verification Plan and add related details

7.7 Week of 5/27-6/2/2024

This week's focus is on completing the final Milestone #5 deliverables and final project presentation for submission on 5/30 AND/OR 6/4. Here are the team responsibilities for the week:

Nick Allmeyer:

- Create UVM environment
- Ensure all testcases are completed and reflected in the coverage reports
- Support others when needed

Alexander Maso:

- Polish UVM testbench and ensure UVM architecture is respected
- Create scenarios of bug-injection and verify results
- Support others when needed

Ahliah Nordstrom:

- Update version control sites, Verification Plan, and specification documents
- Begin final report and presentation formatting
- Create scenarios of bug-injection and verify results
- Support others when needed

7.8 Week of 6/3-6/9/2024

This week's focus is on preparing for our final presentation on 6/4 OR 6/6. Here are the team responsibilities for the week:

Nick Allmeyer:

- Finish final report and presentation formatting
- Support others when needed

Alexander Maso:

- Finish final report and presentation formatting
- Support others when needed

Ahliah Nordstrom:

- Update version control sites, Verification Plan, and specification documents
- Finish final report and presentation formatting
- Support others when needed

References

[1] “Crossing clock domains with an Asynchronous FIFO,” zipcpu.com. https://zipcpu.com/blog/2018/07/06/afifo.html. [Accessed April 20, 2024]

[2] C. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers.* March 2002, Vol. 281.

[3] R. Salemi, “The UVM Primer: An Introduction to the Universal Verification Methodology”. Boston: Boston Light Press, 2013. Accessed: Apr. 1, 2024. [Online].

[3] B. Wile and J. Gross and W. Roesner, “Comprehensive Functional Verification: The Complete Industry Cycle”. San Francisco, CA, USA: Morgan Kaufmann, 2005. Accessed: Apr. 1, 2024. [Online].