Portland State UNIVERSITY

IMPLEMENTATION AND VERIFICATION OF ASYNCHRONOUS FIFO USING BOTH CLASS-BASED AND UVM METHODOLOGIES

DESIGN SPECIFICATION

Fundamentals of Pre-Si Validation

ECE593

Manoj Bontalakoti

Balasainath Reddy Yelampalle

Nikhilesh Devarasetty

Tanvitha Koganti

Table of Contents

1. Introduction

1.1 Purpose

This document provides a comprehensive specification for the Asynchronous First-In-First-Out (FIFO) memory module, Version 1.0. It aims to present a detailed overview of the module's architecture, covering functional aspects, interface requirements, and verification aspects. This document serves as a guide for system architects, design engineers, and verification engineers to assist in the creation, integration, and use of the FIFO in digital systems. The primary goal is to describe how the FIFO manages asynchronous data transfer and synchronization between various clock domains, enhancing data reliability and efficiency.

1.2 Document Conventions

To improve readability and comprehension, this document uses the following conventions:

- **Bold Text:** Highlights important terms, headings, and points.
- *Italic Text:* Emphasizes key concepts.
- `Monospaced Font:` Indicates code snippets, programming language keywords, and command-line examples.
- "Quotation Marks": Used for direct quotes from specifications, standards, or important statements.

1.3 Intended Audience and Reading Suggestions

This document is primarily intended for system architects, design and verification engineers, and technical project managers. It provides comprehensive technical insights suitable for both detailed technical analysis and complete project planning.

1.4 Product Scope

The Asynchronous FIFO memory module improves data flow in complex digital systems by offering buffering and smooth data synchronization across several clock domains. It is essential for maintaining data integrity and reducing latency in systems such as CPUs, network chips, SoCs, and communication interfaces.

1.5 References

- Simulation and Synthesis Techniques for Asynchronous FIFO Design
- Why Do We Use a Gray Encoded Signal in Asynchronous FIFO?

---

2. Overall Description

2.1 Product Perspective

The asynchronous FIFO memory module is designed for integration into complex digital systems like SoCs, CPUs, and network chips. It ensures continuous data flow and integrity by synchronizing data transmission between subsystems with varying operating frequencies.

2.2 Product Functions

The asynchronous FIFO memory module performs the following tasks:

- **Data Buffering:** Temporarily stores data to balance quick producers and slower consumers.
- **Clock Domain Synchronization:** Facilitates data flow between components with different clock domains.
- **Data Integrity Maintenance:** Ensures accuracy of data throughout transmission.
- **Flow Control Management:** Prevents data overflow or underflow by regulating data flow.
- **Status Indication:** Provides signals for system operations (e.g., 'FIFO full', 'FIFO empty').
- **Flexible Data Handling:** Adapts to various data sizes and formats.
- **Interface Compatibility:** Easily integrates with common interfaces in SoCs, CPUs, and network chips.

2.3 User Classes and Characteristics

Users include digital system architects, hardware designers, firmware/software developers, system integrators, embedded system designers, research teams, academic institutions, and project managers. The module meets specific needs in digital system architecture with features like clock synchronization, flexible data handling, binary-to-gray code translation, and interface compatibility.

2.4 Tools and Software

The module communicates with various hardware and software components, operating in diverse digital system environments. It is compatible with popular digital design tools like Synopsys Design Compiler, Cadence Virtuoso, and Xilinx Vivado.

2.5 Design and Implementation Constraints

- **Clock Frequency Compatibility:** Designed to function with a 50% duty cycle on both producer (wclk) and consumer (rclk) clocks at different frequencies.
- **Data Width Compatibility:** Configurable data width, ensuring it matches specific application requirements to prevent data truncation or overflow.
- **Clock Domain Synchronization:** Ensures correct clock domain crossings to avoid data synchronization issues.
- **Data Handling Limitations:** Considers data rates and flow control mechanisms to prevent FIFO overflow or data loss.

2.6 Assumptions and Dependencies

- **Assumptions:**
  - Users provide stable clock signals (wclk and rclk) at designated frequencies and duty cycles.
  - Configuration parameters like data width, clock frequencies, and operation modes are specified by users.
  - Appropriate flow control mechanisms are implemented by the user to prevent data overflow and underflow.
- **Dependencies:**
  - Requires appropriate clock sources that meet design requirements.
  - Utilizes digital design tools and verification tools like Questa Sim and EDA software for integration.
  - May require FPGAs for design integration and verification.
  - Requires proper clock domain crossing methods to synchronize data.

---

3. Data Flow and Working

3.1 Dataflow and Clock Edge Triggering

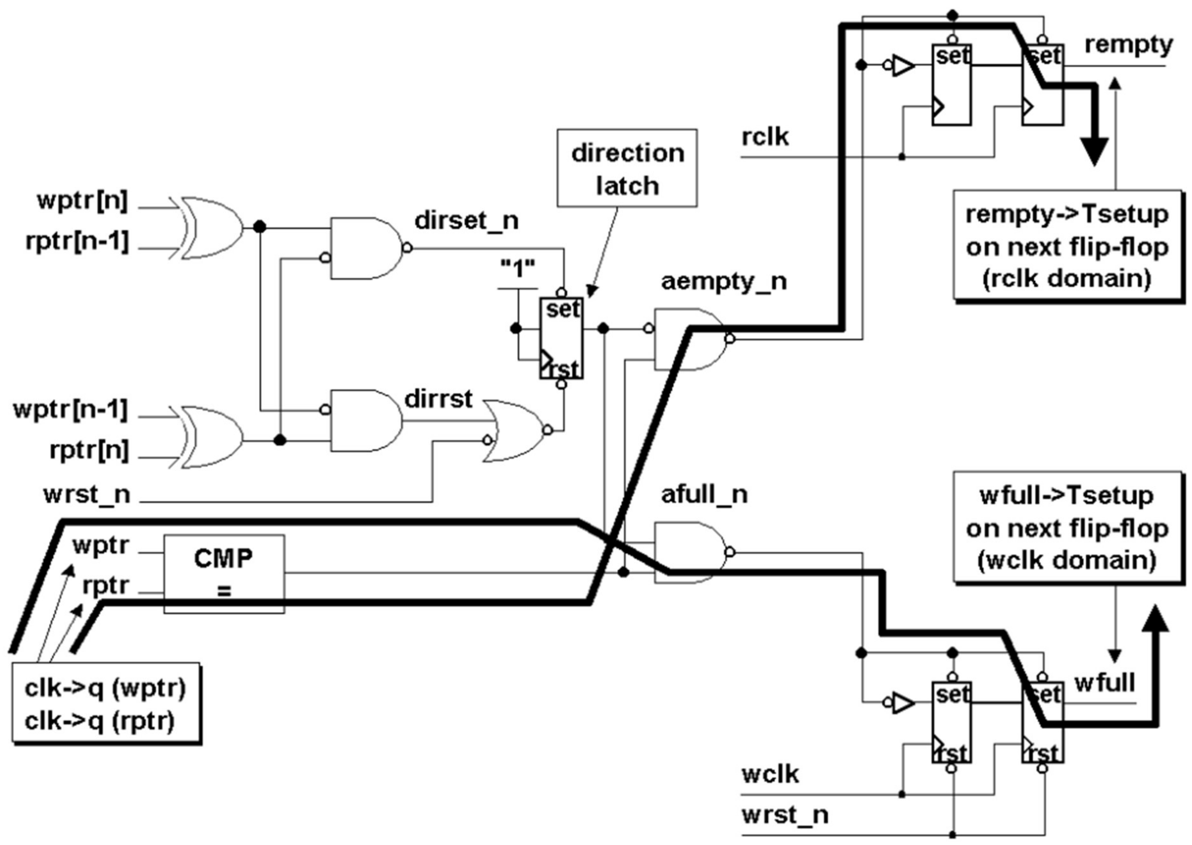
- **Write Operation:** Data entries are written to the FIFO on the positive edge of the producer clock (wclk), ensuring stable and predictable data latching.
- **Read Operation:** Data is read from the FIFO on the positive edge of the consumer clock (rclk), minimizing read latency and ensuring data integrity.

3.2 Clock Domain Interaction and Synchronization

- **Clock Domains:** The FIFO operates on two distinct clock domains, wclk for writing and rclk for reading, both running at 500 MHz.
- **Synchronizers:** Necessary for safely transferring read and write pointers between wclk and rclk domains, preventing issues such as metastability.

3.3 Status Indicators

- **FIFO Empty:** Asserted when the read pointer equals the write pointer, indicating no data is present.
- **FIFO Full:** Asserted when the write pointer is about to overlap the read pointer, indicating no space left.
- **Almost-Empty Warning:** Asserted when the FIFO is nearing empty status.
- **Almost-Full Warning:** Asserted when the FIFO is nearing its full capacity.



3.4 Reset Functionality

- **Reset Mechanism:** A reset input initializes the FIFO state, clearing its content, resetting read/write pointers, and resetting all status flags (Full, Empty, Almost Full, Almost Empty).
- **Asynchronous Reset:** Allows immediate response, independent of clock cycles.

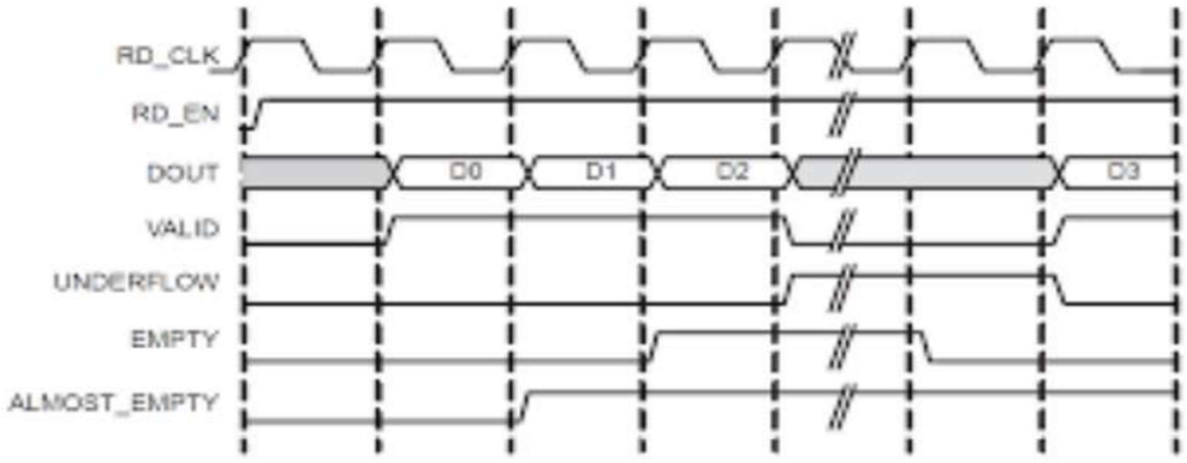
3.5 Counter Design

- **Dual Binary-Gray Counter System:** Uses Gray code for pointer comparisons and status flag generation to ensure stability across clock cycles.

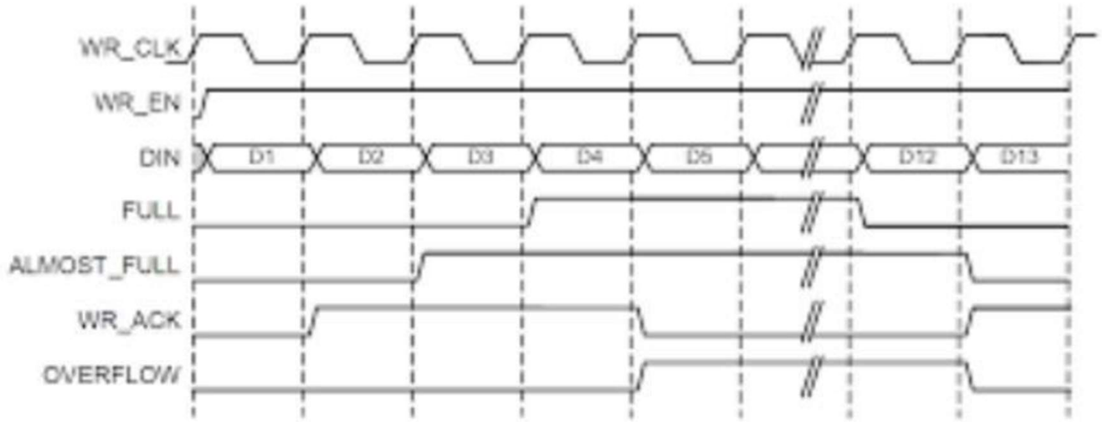
3.6 Timing Diagram

- Illustrates the timing of read and write operations, showing the synchronization between wclk and rclk.

Read operation



Write Operation



---

4. External Interface Requirements

4.1 Hardware Interfaces

Data Input and Output:

- **Data In (DI):** Allows external modules to write data into the FIFO, following the specified data width configuration.
- **Data Out (DO):** Allows external modules to retrieve data from the FIFO, matching the configured data width.

Clock Signals:

- **wclk:** Powers write operations, operating at 1 GHz with a 50% duty cycle.
- **rclk:** Regulates read operations, operating at 500 MHz with a 50% duty cycle.

Control Signals:

- **Write Enable (WE):** Activates or deactivates write operations.
- **Read Enable (RE):** Controls data reading from the FIFO.
- **Clear/Reset (CLR):** Resets the FIFO, clearing its contents and restoring it to an empty state.

Status Signals:

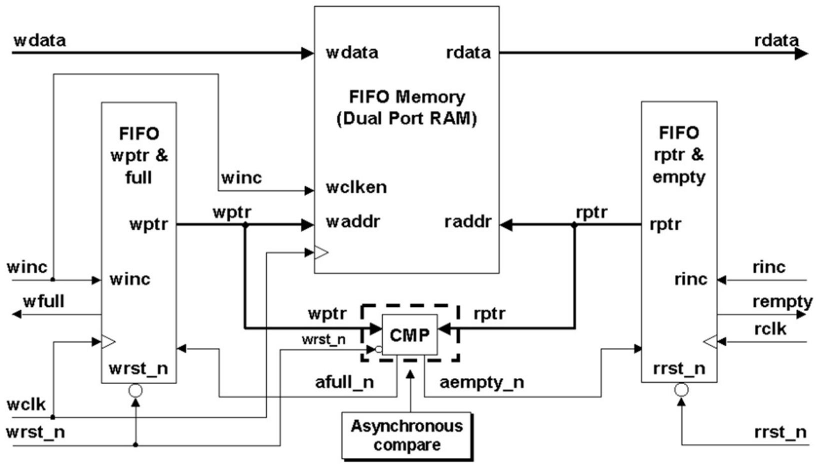
- **FIFO Full (FULL):** Indicates the FIFO is full.
- **FIFO Empty (EMPTY):** Indicates the FIFO is empty.

Binary-to-Gray Code Conversion:

- If configured for binary-to-gray code conversion, includes additional logic for this function.

Synchronization Logic:

- Maintains data synchronization between wclk and rclk as per design criteria.



4.2 Software Interfaces

Digital Design and Verification Tools:

- **Questa Sim:** Used for design, compilation, and verification of the FIFO module.
- **Xilinx Vivado:** Integrates the FIFO module into FPGA platforms, offering synthesis, implementation, and verification capabilities.

Clock Domain Crossing Techniques:

- Essential for data synchronization between multiple clock domains, utilizing appropriate techniques and tools.

---

5. Product Features

5.1 FIFO Memory

The main component of the design, storing data written by the producer clock (wclk at 1 GHz) and read by the consumer clock (rclk at 500 MHz). The FIFO depth accommodates a maximum burst size of 200 writes, maintaining data integrity across different time domains.

5.2 Synchronizers

Preserve data integrity between different clock domains by synchronizing read and write pointers between producer and consumer clocks.

5.3 Write and Read Control Blocks

Control the timing of write and read operations, ensuring there are no data collisions and that the FIFO operates without overrun or underrun errors.

5.4 Binary and Gray Counter

Manage read and write pointers using binary counters while using Gray coding for secure pointer transfer between clock domains, reducing metastability risks.

5.5 Interface Logic

Ensures compatibility and smooth data flow, considering high frequency and specific timing requirements.

---

6. Logic Design

6.1 Directory Structure

1. **Top.sv:** Contains instantiation of the producer module, consumer module, control unit, and FIFO using interfaces and modports.
2. **Consumer.sv:** Manages read operations in the FIFO with control unit output.

3. **Producer.sv:** Manages write operations in the FIFO with control unit output.
4. **Control Unit:** Provides Full and Empty conditions to the producer and consumer.
5. **Asynchronous_fifo.sv:** Defines FIFO operations for writing and reading.
6. **interface_fifo.sv:** Contains interfaces and modports for submodules.
7. **Asynchronous_fifo_tb.sv:** Conventional test bench using tasks and threads.

6.2 Design Modules

- **Top.sv**
- **Interface.sv**
- **Asynchronous_fifo.sv**
- **Producer.sv**
- **Consumer.sv**
- **Control Unit.sv**
- **Asynchronous_tb.sv**

6.3 System Verilog Abstraction Features Used

- Data Types: Utilizes 4-state logic data type and 2-state bit data type.
- `always_ff`, `always_comb`, `always_latch`
- Interfaces and Modports
- Tasks and Functions

6.4 Simulation, Tools, Directory Structure

- Used Questa-sim for simulation.

---

7. Verification

7.1 Testbench Style

- Conventional Testbench using tasks and functions.

7.2 Testing Strategies

- Dynamic simulation to examine the asynchronous FIFO design under various operating scenarios, including clock domain interactions and real-time data flow.

7.3 Test Case Scenarios

- **Test Full:** Writes data to all FIFO locations until full, verifying the full flag and FIFO behavior when attempting further writes.
- **Test Empty:** Reads data until FIFO is empty, verifying the empty flag and FIFO behavior when attempting further reads.
- **Test Full Error:** Verifies behavior when attempting to write data to a full FIFO.
- **Test Empty Error:** Verifies behavior when attempting to read data from an empty FIFO.
- **Test Concurrent Write-Read:** Verifies concurrent writing and reading operations, ensuring no race conditions or synchronization issues.

7.4 Code Coverage

- **Statement Coverage:** Ensures every line of code is executed at least once.
- **Branch Coverage:** Ensures both branches of all decision points are tested.
- **Path Coverage:** Ensures every route through the control flow is traversed.
- **Function and Procedure Coverage:** Ensures thorough testing of every process and function, including boundary cases and various usage scenarios.

---

8. References

- "Simulation and Synthesis Techniques for Asynchronous FIFO Design" by Clifford E. Cummings, Sunburst Design, Inc. Sunburst Design Paper