# High Level Design Specification (HLDS)

**for**

# <Asynchronous FIFO Design Specification>

**Version 1.0**

**Prepared by <Ashwinkumar Sivakumar, Mahidhar Regalla, Monica Pinnamaneni>**

**<ECE-593: Fundamentals of Pre-Silicon Validation – Venkatesh Patil>**

**<1/11/2024>**

# Table of Contents

## Contents

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| Ashwinshivakumar Mahidhar Regalla Monica Pinnamaneni | 1/12/2024 | Base documentation from section 1 to 4 | 1.0 |
| Ashwinshivakumar Mahidhar Regalla Monica Pinnamaneni | 02/01/2024 | Revised documentation from section 1 to 8 | 1.1 |

# 1. Introduction

## 1.1 Purpose

*This document provides a detailed specification for the Asynchronous First-In-First-Out (FIFO) memory module, Revision 1.0. It is directed to provide a clear and detailed overview of the module's design, including architectural features, functionality, interface specifications and verification aspects. The document is intended to serve as a valuable resource for design and verification engineers, as well as system architects, guiding them through the development, integration, and application of the FIFO in digital systems. The purpose of this document is to outline the FIFO's capabilities in managing asynchronous data transfer and synchronization between different clock domains, thereby improving the efficiency and reliability of digital data*

## 1.2 Document Conventions

*In this High-Level Design Specification (HLDS) document for the Asynchronous First-In-First-Out (FIFO) memory module, Revision 1.0, specific typographical and formatting conventions have been employed to enhance readability and understanding. These conventions are as follows:*

***Bold Text:*** *Used for emphasizing critical terms, headings, and key points within the document. It is also used for highlighting important features and names of interfaces or components.*

***Monospaced Font:*** *Utilized for code snippets, programming language keywords, and command-line examples to distinguish them from regular text.*

***"Quotation Marks":*** *Applied when directly quoting specifications, standards, or importantstatements from referenced documents.*

***Bullet Points and Numbered Lists:*** *Used for organizing information logically, especially insections detailing features, requirements, and step-by-step instructions.*

***Underlining:*** *Reserved for hyperlinks and interactive elements in electronic versions of the document, providing direct access to referenced documents or external resources.*

***Color Highlights:*** *Used sparingly, primarily to draw attention to critical warnings, errors, or crucial notes that require special attention.*

*These conventions have been consistently applied throughout the document to facilitate a uniform and clear understanding of the content, thereby aiding in the document's utility as a comprehensive guide for the intended audience.*

## 1.3 Intended Audience and Reading Suggestions

*This High-Level Design Specification (HLDS) for the Asynchronous FIFO memory module is aimed primarily at design and verification engineers, system architects, and technical project managers, providing detailed technical insights for development and integration. The document is designed to accommodate both detailed technical analysis and comprehensive project planning requirements.*

## 1.4 Product Scope

*The Asynchronous FIFO memory module is aimed to enhance data flow in complex digital systems by providing smooth data synchronization and buffering across multiple clock domains. Its integration is critical for improving data integrity and reducing latency in larger systems such as SoCs, CPUs, network chips, and communication interfaces. The module's ability to adapt to different data rates, as well as its robust performance, are consistent with the goals of increasing efficiency and reliability in high-speed digital environments. This makes it an important component in advanced digital architectures, significantly improving their overall effectiveness and scalability.*

## 1.5 References

1. *S. Cummings, "FIFOs: Fast, predictable, and deep," in Proceedings of SNUG, 2002. [Online]. Available: http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf.*
2. *S. Cummings, "FIFOs: Fast, predictable, and deep (Part II)," in Proceedings of SNUG, 2002. [Online]. Available: http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf.*
3. *Putta Satish, "FIFO Depth Calculation Made Easy," [Online]. Available: https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf.*
4. *A. Author et al., "Title of the Paper," in Proceedings of the Conference, 2015, pp. 123-456. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7237325.*
5. *M. Last Name et al., "Designing Asynchronous FIFO," [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/56108360/EC109-libre.pdf.*
6. *A. Author et al., "Title of the Paper," in Proceedings of the Conference, 2011, pp. 789-012. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6041338*
7. *"Youtube," [Online]. Available: https://www.youtube.com/watch?v=UNoCDY3pFh0*
8. *Open AI, "Chat GPT," [Online].*

# 2. Overall Description

## 2.1 Product Perspective

*The Asynchronous FIFO memory module is a vital component specifically designed for integration into complex digital systems such as System-on-Chips (SoCs), CPUs, and network chips. It playsa key role in synchronizing data transfer between subsystems that operate at similar frequencies but with different data processing rates, ensuring seamless data flow and integrity.*

*The origin of this module is rooted in the need for improved data transfer mechanisms in systems where traditional synchronous methods were insufficient, especially in scenarios involving high data throughput and minimal latency.*

***Major components of the overall system:***

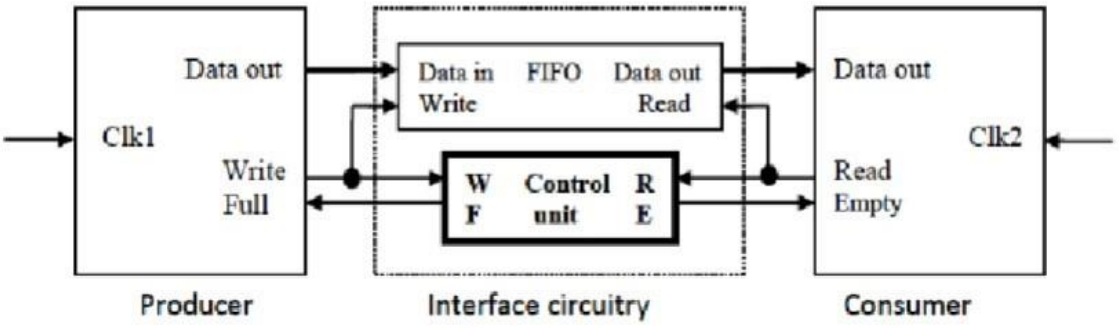



*Fig 1 Components of the FIFO*

## 2.2 Product Functions

*The Asynchronous FIFO memory module serves essential functions in digital systems:*

- ***Data Buffering**: Temporarily stores data, balancing fast producers and slower consumers.*
- ***Clock Domain Synchronization**: Facilitates data transfer between components with different clock domains, ensuring data integrity.*
- ***Binary to gray code conversion**: Reduce the chance of erroneous readings ● **Dat aIntegrity Maintenance**: Ensures data remains accurate during transfer.*
- ***Flow Control Management**: Prevents data overflow or underflow by controlling data flow.*
- ***Status Indication**: Provides signals ('FIFO full,' 'FIFO empty,' etc.) for system actions.*

- ***Flexible Data Handling****: Adapts to varying data sizes and formats.*
- ***Interface Compatibility****: Integrates seamlessly with SoCs, CPUs, and network chips through standard interfaces.*

## 2.3 User Classes and Characteristics

*The Asynchronous FIFO memory module supports a wide range of user classes, including digital system architects, hardware designers, firmware/software developers, system integrators, embedded system designers, research teams, academic institutions, and project managers/operators. It addresses specific needs in digital system design by providing features such as clock synchronization, flexible data handling, binary-to-gray code conversion, and interface compatibility. These characteristics offer practical solutions to a variety of design and integration challenges, making it useful for professionals and researchers.*

## 2.4 Tools and Software

*The Asynchronous FIFO memory module operates in digital system environments and interactswith various hardware and software components. Its environment includes:*

***Hardware Platform:*** *The module is intended to work within digital systems such as System-on-Chip (SoCs), CPUs, and network chips, where it serves as an important data manager.*

***Operating Systems:*** *The FIFO module is platform-agnostic and can run in a variety of environments. It does not rely on a particular operating system.*

***Digital Design Tools:*** *Users can use digital design tools and Electronic Design Automation (EDA) software to integrate the FIFO module into their hardware designs. Popular tools such as Xilinx Vivado, Cadence Virtuoso, and Synopsys Design Compiler may be utilized.*

***Firmware and Software:*** *The FIFO module's integration with firmware and software components is determined by the digital system's specific requirements. It is compatible with firmware and software applications designed for the target system.*

## 2.5 Design and Implementation Constraints

*Asynchronous FIFO memory module design and implementation constraints, considering the specified design specifications:*

***Clock Frequency Compatibility****: The module is designed to operate under the conditions of a producer clock (clk1) frequency of 500 MHz and a consumer clock (clk2) frequency of 500 MHz, both with a 50% duty cycle.*

***Data Width Compatibility****: The module's data width is adaptable, but it should align with the data width requirements of your specific application. Configure the FIFO for the appropriate data width to avoid data truncation or overflow issues, considering the maximum write burst size of 300 specified in the design specifications.*

***Clock Domain Synchronization****: While the module facilitates clock domain synchronization,ensure that proper clock domain crossings are implemented in your designs to prevent data synchronization issues. The ideal cycle between successive writes and reads should be considered (2 and 5 cycles, respectively) as per the design specifications.*

***Data Handling Limitations:*** *Given the specified conditions, be mindful of data rates and flow control mechanisms. Excessive data rates without proper flow control may lead to data loss or FIFO overflow. The maximum writes burst size and ideal cycles between successive writes and reads should be taken into account.*

***Interface Compatibility:*** *Verify that your system's interfaces align with the module's requirements for the specified conditions, including clock frequencies and data widths, as outlined in the design specifications.*

***Binary-to-Gray Code Conversion:*** *If the module is used for binary-to-Gray code conversion, ensure that your application requires this feature and configure the module accordingly, taking into consideration the design specifications.*

## 2.6 Assumptions and Dependencies

### 2.6.1 Assumptions

***Clock Signals****: Users are expected to provide stable clock signals (clk1 and clk2) at specified frequencies and duty cycles in accordance with design specifications.*

***Configuration****: Users must set the FIFO module's data width, clock frequencies, and operating parameters in accordance with design specifications.*

***Flow Control****: The user is responsible for implementing proper flow control mechanisms to prevent data overflow and underflow.*

***Binary-to-Gray Code Conversion:*** *If this feature is enabled, users assume alignment with their application requirements and perform any necessary configurations.*

### 2.6.2 Dependencies

***Clock Sources:*** *Proper clock sources that meet specified requirements are critical for module operation.*

***Tools:*** *Users may need digital design tools and Verification tools like Questa sim and EDAsoftware for integration.*

***Hardware:*** *Users may need FPGAs for integrating and testing the design.*

***Clock Domain Crossing:*** *Proper clock domain crossing techniques are required for data synchronization.*

# 3. Data flow and Working

## 3.1 Dataflow and Clock Edge Triggering

***Write Operation:*** *Data entries into the FIFO are written on the positive edge (posedge) of the producer clock (clk1). This ensures that data is latched consistently at the highest point of the clock cycle, providing stability and predictability in high-speed environments.*
***Read Operation:*** *Data is read from the FIFO on the positive edge (posedge) of the consumer clock (clk2). This alignment with the rising edge of the clock ensures data integrity and minimizes read latency.*

## 3.2 Clock Domain Interaction and Synchronization

***Clock Domains:*** *The FIFO operates across two different clock domains, clk1 for writing and clk2 for reading, both operating at 250MHZ and 100MHZ.*

***Synchronizers:*** *Given the asynchronous nature of the FIFO, synchronizers are essential to safely pass the read and write pointers between clk1 and clk2 domains. This prevents issues like metastability which can occur when signals are transferred between asynchronous clock domains.*
***Number of Synchronizers:*** *Typically, a chain of two or three flip-flops is used as a synchronizer for each signal crossing the clock domains, ensuring reliable signal transfer.*

## 3.3 Status Indicators and Protocols

***ÏÏÏO Empīy:*** *Asseíted when the íead pointeí equals the wíite pointeí, indicating no data is píesent.*

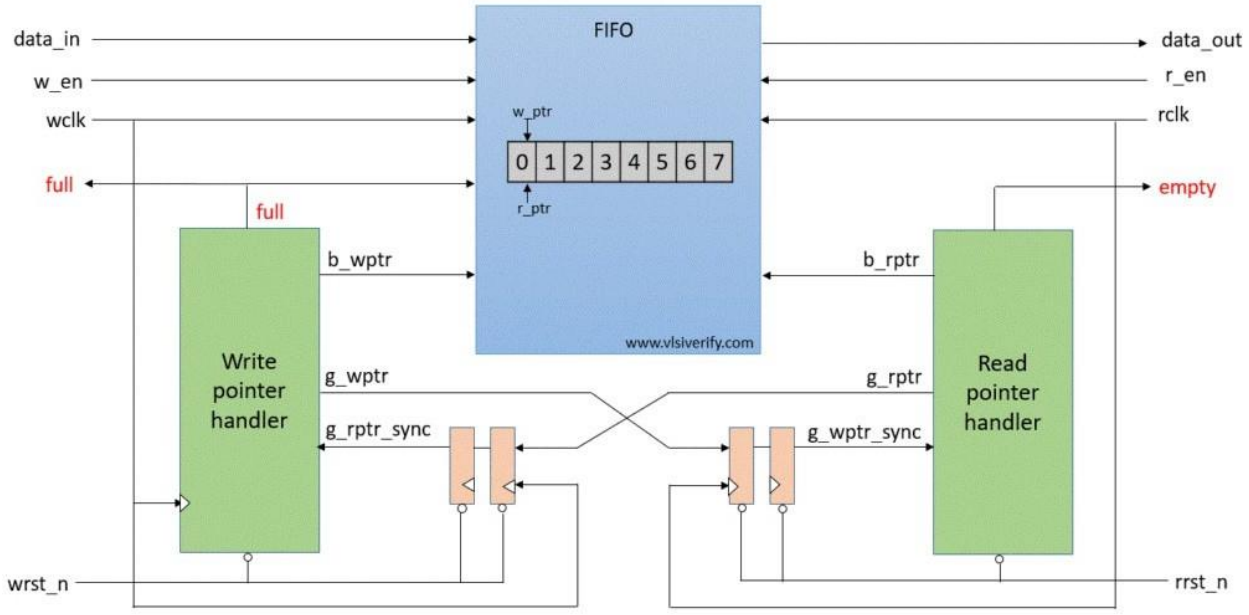
***ÏÏÏO Ïull:*** *Asseíted when the wíite pointeí is about to oveílap the íead pointeí, indicating nospace left.*

***Almosī-Empīy Waíning:*** *A thíeshold is set below which the Almost Empty flag is asseíted, indicating the FIFO is neaíing empty status.*

***Almosī-Ïull Waíning:*** *Similaí to Almost Empty, a thíeshold is set befoíe the FIFO is full, beyond which the Almost Full flag is asseíted, indicating the FIFO is neaíing its full capacity.*

### 3.3.1 Read and write pointer handler

*Ïhe Wíite and Read Pointeí Handleís aíe integíal featuíes of an Asynchíonous FIFO, each seíving a vital function in managing data flow within the FIFO.*




**Asynchronous FIFO**

#### *Wíiīe poinīeí Handleí*

*Ïhe Wíite Pointeí Handleí is íesponsible foí tíacking wheíe new data is wíitten in the FIFO memoíy. It advances sequentially with each wíite opeíation and incoípoíates logic to identify when the FIFO is full, píeventing data oveíwíiting. Ïhis pointeí is cíucial in managing the FIFO's status, paíticulaíly in ensuíing that no new data is wíitten when the FIFO is at capacity.Additionally, in asynchíonous enviíonments, it includes synchíonization mechanisms to safely tíansfeí the wíite pointeí infoímation acíoss diffeíent clock domains, a cíitical aspect to avoid data coííuption oí loss.*

#### *Read poinīeí handleí*

*Ïhe Read Pointeí Handleí, on the otheí hand, tíacks the memoíy location fíom wheíe data should be íead, ensuíing the FIFO's Fiíst-In-Fiíst-Out opeíation píinciple. It incíements afteí each íead opeíation and includes logic to detect an empty FIFO, píeventing undeíflow and maintaining data integíity. Like the wíite pointeí, the íead pointeí also íesets to the staít position upon initialization and includes wíap-aíound logic foí continuous data access. In asynchíonous FIFOs, it too íequiíes synchíonization when cíossing clock domains, a featuíe essential to maintaining consistent FIFO status and ensuíing seamless data íead opeíations.*

## 3.4 Clear/Reset Functionality

***Reset Mechanism:*** *A reset input is provided to initialize the FIFO state. When activated, it clears the FIFO content, resets the read/write pointers, and resets all status flags (Full, Empty, Almost Full, Almost Empty).*
***Asynchronous Reset:*** *The reset is designed to be asynchronous to allow immediate response, independent of clock cycles.*

## 3.5 Counter Design

***Type of Counter:*** *The design utilizes a dual Binary-Gray counter system. The binary counter manages the read and write pointers internally, while the Gray code representation is used for pointer comparisons and status flag generation to ensure one-bit change stability across clock domains.*

## 3.6 Timing Diagram

**Read operation**

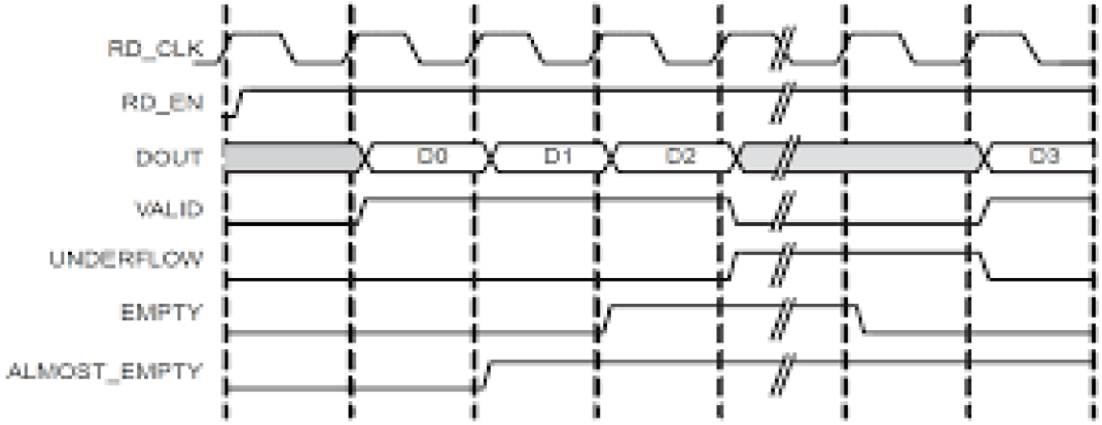



Fig 2 Read timing diagram

**Write operation**

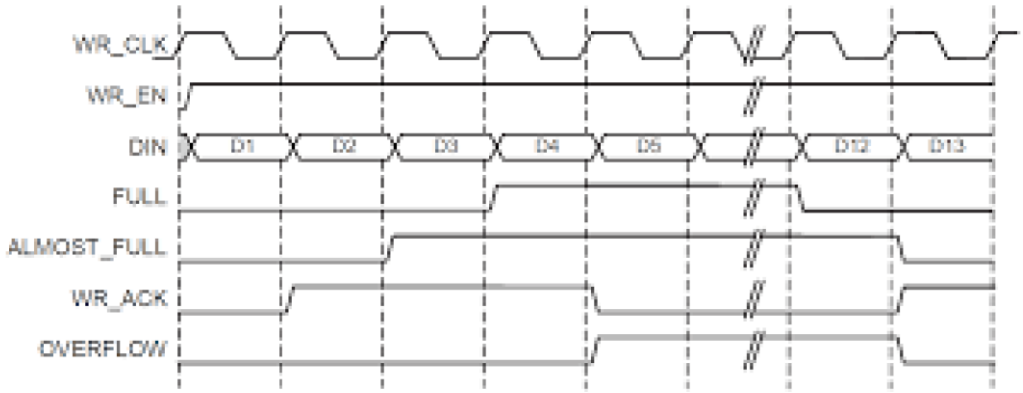



Fig 3 Write timing diagram

# 4. External Interface Requirements

## 4.1 Hardware Interfaces

*In the hardware interfaces section, we will describe the logical characteristics of the Asynchronous FIFO memory module and its interactions with interfacing modules. Here's an overview of the hardware interfaces and key aspects:*

### 4.1.1 Data Input and Output

***Data In (DI):*** *This interface allows external modules to write data into the FIFO. It should adhere to the data width configuration specified in the design specifications.*

***Data Out (DO):*** *This interface enables the retrieval of data from the FIFO by external modules. The data width should match the configuration set during initialization.*

### 4.1.2 Clock Signals

***clk1:*** *This clock signal, with a frequency of 250 MHz and a 50% duty cycle (as specified in the design), drives the write operations of the FIFO.* ***clk2:*** *This clock signal, also with a frequency of 100 MHz and a 50% duty cycle, controls the read operations of the FIFO.*

### 4.1.3 Control Signals

***Write Enable (WE):*** *External modules use this signal to enable or disable write operations to the FIFO.*
***Read Enable (RE):*** *This signal controls the reading of data from the FIFO.*
***Clear/Reset (CLR):*** *The CLR signal is used to reset the FIFO, clearing its contents and returning itto an empty state as per design specifications.*

### 4.1.4 Status Signals

***FIFO Full (FULL):*** *Indicates that the FIFO is full and can no longer accept write operations.*
***FIFO Empty (EMPTY):*** *Signifies that the FIFO is empty and cannot be read until new data is written.*

### 4.1.5 Binary-to-Gray Code Conversion

*If configured for binary-to-Gray code conversion, the FIFO may have additional logic for this purpose. External modules can interface with this feature as needed.*

### 4.1.6 Synchronization Logic

*The FIFO includes internal logic for clock domain crossing to ensure data synchronization between the two clock domains (clk1 and clk2) as per design specifications.*

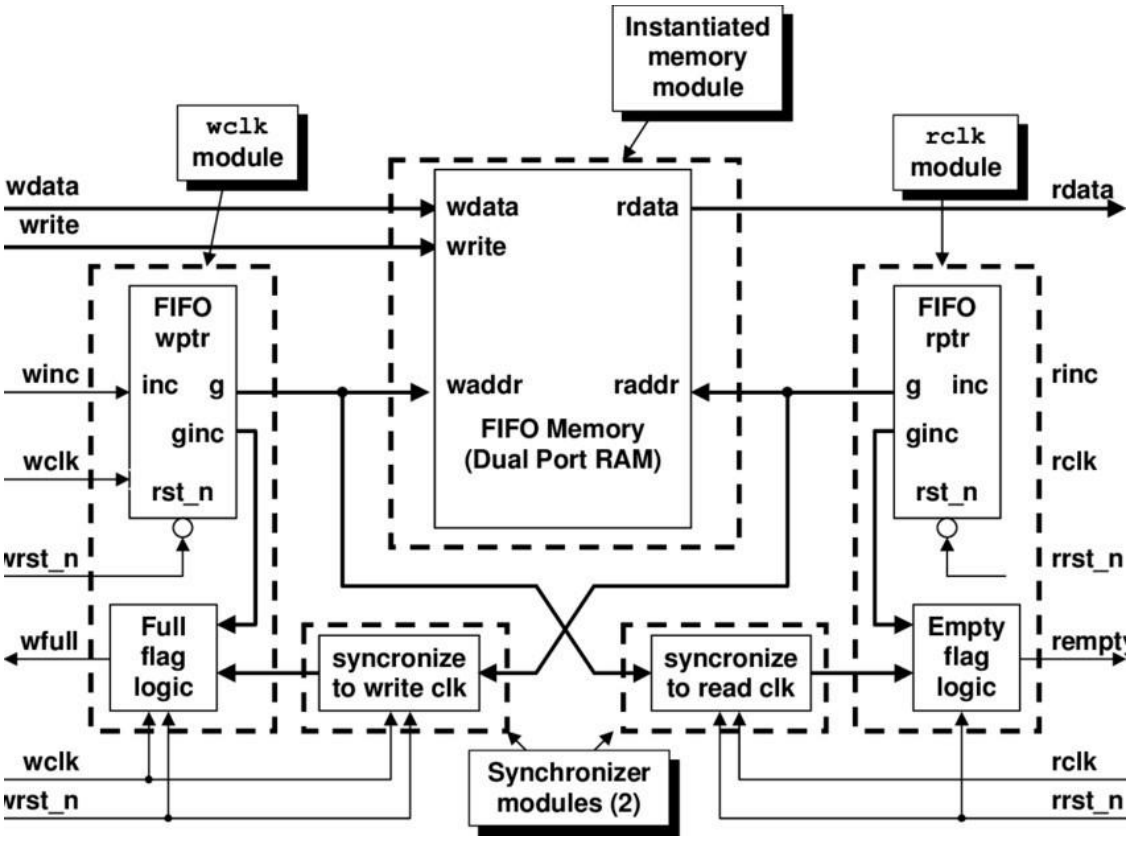



*Fig 4 Detailed hardware interface blocks*

## 4.2 Software Interfaces

*The Asynchronous FIFO memory module primarily operates within digital system environments and interfaces with various software components during its integration and verification processes. Here are the key software interfaces and dependencies:*

### 4.2.1 Digital Design and Verification Tools

***QuestaSim:*** *Users can employ QuestaSim, a powerful simulation and verification tool, for designing, compiling, and verifying the FIFO module. QuestaSim facilitates hardware description language (HDL) simulation, ensuring the correctness and functionality of the design.*
***Xilinx Vivado:*** *For FPGA-based designs, Xilinx Vivado is a valuable tool for integrating the FIFO module into FPGA platforms. Vivado provides synthesis, implementation, and verification capabilities to ensure successful FPGA integration.*

### 4.2.2 Operating Systems

*The FIFO module itself does not have dependencies on specific operating systems. It is generally platform-agnostic and can operate in various environments.*

### 4.2.3 Clock Domain Crossing Techniques

*While not software in the traditional sense, clock domain crossing techniques play a crucial role in the synchronization of data across different clock domains. Users may employ methodologies and tools to implement proper clock domain crossing techniques in their designs.*

# 5. Product Features

*The asynchronous FIFO design is optimized for high-speed operation , with efficient synchronization, buffering, and control mechanisms to handle burst data transfers and different idlecycle requirements for reads and writes.*

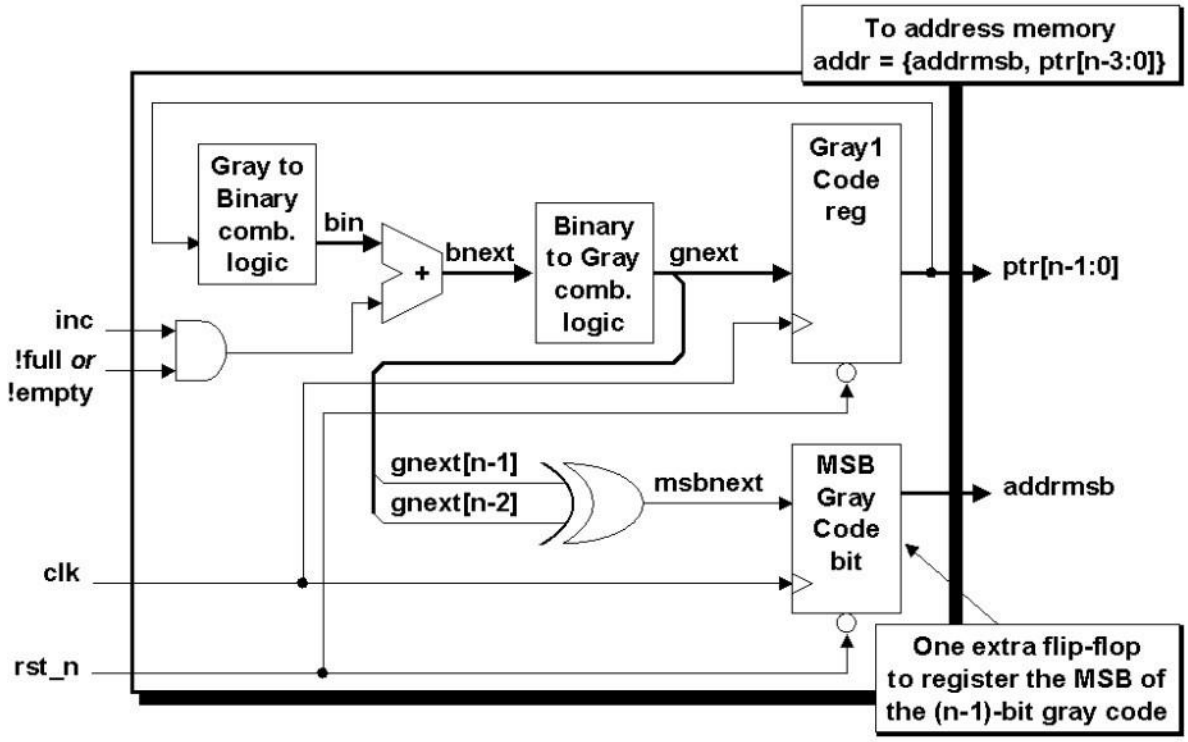
## 5.1 FIFO Memory

*This is the central component of the design. It stores data written by the producer clock (clk1 at 250 MHz) and makes it available for the consumer clock (clk2 at 100 MHz). The memory is designed to handle a maximum burst size of 150 writes, with the FIFO depth tailored accordingly. The FIFO ensures data integrity despite the different clock domains for producer and consumer.*

## 5.2 Binary and Gray counter

*The design incorporates binary counters for managing write and read pointers. However, to safely transfer these pointers between different clock domains (250 MHz and 100 MHZ with a 50% duty cycle for both producer and consumer), Gray coding is used. This coding ensures that only one-bit changes at a time during pointer updates, significantly reducing the risk of metastability during clock domain crossing.The style #1 Gray code counter assumes that the outputs of the register bits are the Gray codevalue itself (ptr, either wptr or rptr). The Gray code outputs are then passed to a Gray-to-binary converter (bin), which is passed to a conditional binary-value incrementer to generate the next- binary-count-value (bnext), which is passed to a binary-to-Gray converter that generates the next-Gray-count-value (gnext), which is passed to the register inputs. The top half of the Figure block diagram shows the described logic flow while the bottom half shows logic related to the second Gray code counter as described in the next section.*

*Fig 5 - Dual n-bit Gray code counter block diagram - style #1*




## 5.3 Synchronizers

*Critical for maintaining data integrity across different clock domains. These components synchronize the read and write pointers between the producer and consumer clocks. The design typically involves a multi-stage flip-flop chain, ensuring reliable pointer updates even in the presence of clock skew and varying duty cycles.*

## 5.4 Write and Read control Blocks

*These blocks manage the timing of writes and reads. The design enforces a minimum gap of 2 idle cycles between successive writes and 5 idle cycles between successive reads. This is crucial for preventing data collision and ensuring the FIFO operates without overrun or underrun errors.*

## 5.5 Status indicator

*To provide additional functionality, the FIFO includes flags such as 'Full', 'Empty', 'Almost Full', and 'Almost Empty'. These flags help in managing the data flow, especially in controlling the burst writes and reads. The control logic ensures seamless operation under varying conditions of data availability and space in the FIFO.*

## 5.6 Data Buffering and Flow management

*Given the high operating frequency and the requirement for burst transfers, the FIFO design includes robust data buffering mechanisms. These buffers accommodate the burst writes and ensure smooth data flow even when there's a disparity in read and write speeds.*

### 5.7 Interface Logic

*This includes the logic necessary for interfacing the FIFO with other components in the system. It ensures compatibility and seamless data transfer, considering the high frequency and specific timing requirements.*

# 6. FIFO Depth calculation

***Given specifications:***

*Producer Clk frequency : 250MHz*
*Consumer Clk frequency : 100 MHz*
*Maximum write burst size: 150*
*Duty cycle = 50%*
*No. of Idle cycle between successive writes =0No.*
*of Idle cycle between successive reads = 2*

***Calculations:***

*The no. of idle cycles between two successive reads is 2 clock cycles, it means that after reading One data consumer block is waiting for 2 clock cycles to initiate the next read. So, it can be understood that for every 3 clock cycles, one data is read.*

*clk B = 100MHz*
*clk A = 250MHz*

- *Time required to write one data item: (1/250MHz) = 4 ns, But with a duty cycle of 50% time required is 8ns.*
- *Time required to write all the data in the burst = 150 x 8ns= 1200 ns.*
- *And time required to read one data item = 3 x 1/100 = 30ns but with duty cycle time required is60ns.*
- *In a period of 1200ns no of items can be read is 1200/160 = 20.*
- *The remaining no of bytes to be stored in the FIFO = 150-20 = 130.So, the minimum depth of FIFO should be **130**.*

# 7. Logic Design

## 7.1 <Your work Directory Structure>

*team1_Final>rtl*

- *design.sv*

*team1_Final>sim*

- *testbench.sv*
- *interface.sv*
- *test.sv*
- *env.sv*
- *agent.sv*
- *driver.sv*
- *monitor.sv*
- *sequence.sv*
- *sequencer.sv*
- *sequence_item.sv*
- *scoreboard.sv*

## 7.2 <Design modules>

### 7.2.1 FIFO memory

***Ports Description***

*winc: Write increment signal (input). When high, and wfull is not asserted, data is written into thememory at waddr.*
*wfull: Write full signal (input). When high, it indicates the FIFO memory is full, and no further writeoperations should be attempted.*
*wclk: Write clock signal (input). The rising edge triggers write operations into the FIFO memory.*
*waddr: Write address (input). A binary value specifying the address where the new data should be written. raddr: Read address (input). A binary value specifying the address from which data should be read.*
*wdata: Write data (input). The data to be written into FIFO*
*memory.rdata: Read data (output). The data being read from FIFO*
*memory.*

***Functional Description***

*The FIFO memory operates synchronously for write actions on the rising edge of wclk. A write is executed if winc is asserted and wfull is not, storing wdata into the memory cell addressed by waddr. Reading is asynchronous, with rdata reflecting the contents of the memory cell at raddr instantaneously. The internal memory is a simple array, mem, sized according to DEPTH, which is derived from ADDRSIZE. The module is designed with simplicity in mind, focusing on core FIFO functionalities without handling flag generation for full or empty conditions or reset logic,*

*delegatingthese responsibilities to the encompassing system.*

***Design Considerations***

*The module assumes that the write operations will not be attempted when wfull is asserted, as itdoes not internally manage or detect overflow conditions. Similarly, read operations should be externally managed to prevent underflow. The lack of built-in synchronization for the read operations implies that if used across different clock domains, external synchronization measuresshould be implemented. The current design is optimized for ease of integration and minimal resource utilization, with potential future enhancements including parameter validation, automaticaddress wrapping, and dual-clock domain support.*

### 7.2.2 FIFO synchronization Write to read

***Ports Description***

*rclk: Read domain clock (input). The clock signal used for the read domain, triggering the synchronization process on its rising edge. rrst_n: Read domain reset, active low (input). The reset signal for the read domain. When asserted low, it asynchronously resets the output pointers to zero. wptr: Write pointer (input). A binary vector of width ADDRSIZE + 1bits, representing the current write address in the write domain. rq2_wptr: Synchronized write pointer for the read domain (output). A binary vector of width ADDRSIZE + 1 bits, it isthe write pointer value that has been synchronized to the read domain.*

***Functional Description***

*The sync_w2r module is designed to synchronize a write pointer from the write clock domain to the read clock domain. It employs a two-stage synchronization process to mitigate the risk of metastability when crossing clock domains. The module captures the value of the write pointer (wptr) on the rising edge of the read clock (rclk) and outputs thisvalue as rq2_wptr after two clock cycles of rclk.*

*Upon a reset (when rrst_n is low), the synchronization registers (rq2_wptr, rq1_wptr) are cleared to zero. During normal operation, the write pointer is first latched into rq1_wptr andthen into rq2_wptr, effectively pipelining the synchronization to safely transfer the write pointer into the read domain.*

***Design Considerations***

*The module assumes that wptr is stable before the rising edge of rclk to ensure correct latching. Any changes to wptr should be timed appropriately or held constant during thesynchronization to prevent data corruption.*
*The two-stage pipeline for synchronization helps in reducing the risk of metastability but introduces a two-clock-cycle latency from the write domain to the read domain. This latency*

*should be accounted for in the system-level timing analysis.*
*Reset signal rrst_n is active low and asynchronous, meaning it will reset the output pointers irrespective of the clock signal. Care should be taken to ensure that rrst_n is onlyasserted when it is safe to reset the module without disrupting ongoing operations. The width of wptr and rq2_wptr includes an extra bit to prevent overflow issues when the writepointer wraps around. This extra bit must be managed correctly in the system to maintainthe integrity of the FIFO's status.*
*The module does not include any logic to handle the actual data transfer between domains; it merely synchronizes the write pointer location. Additional logic will be requiredto manage data movement associated with these pointers.*

### 7.2.3 FIFO rptr_empty

***Ports Description***

*rclk: Read domain clock (input). The clock signal used for the read domain, triggering the synchronization process on its rising edge. rrst_n: Read domain reset, active low (input). The reset signal for the read domain. When asserted low, it asynchronously resets the output pointers to zero. rinc: Read increment signal (input). When high, and rempty is notasserted, data is written from the memory at raddr. rq2_wptr: Synchronized write pointer for the read domain (output). A binary vector of width ADDRSIZE + 1 bits, it is the write pointer value that has been synchronized to the read domain. rempty:Read empty signal (input). When high, it indicates the FIFO memory is empty, and no further read operationsshould be attempted. raddr: Read address (input). A binary value specifying the address from which data should be read.*

***Functional Description***

*The module uses a Gray code-style pointer (rptr) for read-addressing.*
*On the rising edge of rclk or when asynchronous reset (rrst_n) is low, rbin and rptr are initialized to zero.The memory read-address pointer (raddr) is derived from the binary version of rbin. rbin is incremented based on rinc and is adjusted based on the not-emptycondition (~rempty). The Gray code version of the next read pointer is computed using rgraynext. The FIFO is considered empty when the next read pointer (rgraynext) matchesthe synchronized write pointer (rq2_wptr). The rempty signal reflects the empty condition.*

***Design Considerations***

*The module assumes external control of read operations to prevent underflow.Lack of built-in synchronization for read operations implies that if used across different clock domains, external synchronization measures should be implemented.The design is optimized for ease of integration and minimal resource utilization.This module works inconjunction with the FIFO memory and helps manage the read pointer and empty condition.*

### 7.2.4 FIFO synchronization Read to Write

***Ports Description***

*wclk: Write clock signal (input). The rising edge triggers write operations into the FIFO memory. wrst_n: Write domain reset, active low (input). The reset signal for the write domain. Whenasserted low, it asynchronously resets the output pointers to zero. wptr: Write pointer (input).A binary vector of width ADDRSIZE + 1 bits, representing the current write address in the write domain. rq2_wptr: Synchronized write pointer for the read domain (output). A binary vector of width ADDRSIZE + 1 bits, it is the write pointer value that has been synchronized to the read domain. waddr: Write address (input). A binary value specifying the address where the new data should be written.*

***Functional Description***

***Functional Description***

*On the rising edge of wclk or when asynchronous reset (wrst_n) is low, rq1_wptr is initialized to zero.The synchronized write pointer for the read domain (rq2_wptr) is updatedby capturing the current value of the write pointer (wptr).This synchronization is performed to align the write pointer with the read clock domain.*

***Design Considerations***

*On the rising edge of wclk or when asynchronous reset (wrst_n) is low, rq1_wptr is initialized to zero.The synchronized write pointer for the read domain (rq2_wptr) is updatedby capturing the current value of the write pointer (wptr).This synchronization is performed to align the write pointer with the read clock domain.*

### 7.2.5 FIFO top

***Ports Description***

*wclk: Write clock signal (input). The rising edge triggers write operations into the FIFO memory. wrst_n: Write domain reset, active low (input). The reset signal for the write domain. Whenasserted low, it asynchronously resets the output pointers to zero.*
*rinc: Read increment signal (input). When high, and rempty is not asserted, data is writtenfrom the memory at raddr.*
*rclk: Read domain clock (input). The clock signal used for the read domain, triggering the synchronization process on its rising edge. rinc: Read increment signal (input). When high, and rempty is not asserted, data is written from the memory at raddr. rrst_n: Read domain reset, active low (input). The reset signal for the read domain. When asserted low, it asynchronously resets the output pointers to zero. wdata: Write data (input). The datato be written into FIFO*

*memory. rdata: Read data (output). The data being read from FIFO memory.*
*rempty:Read empty signal (input). When high, it indicates the FIFO memory is empty, andno further read operations should be attempted.*
*wfull: Write full signal (input). When high, it indicates the FIFO memory is full, and no further write operations should be attempted.* ***Functional Description***

*The module uses a Gray code-style pointer (wptr) for write addressing.*
*On the rising edge of wclk or when asynchronous reset (wrst_n) is low, wbin and wptr are initialized to zero.The memory write-address pointer (waddr) is derived from the binary version of wbin. wbin is incremented based on winc and adjusted based on the not-full condition (~wfull). The Gray code version of the next write pointer is computed using wgraynext. The FIFO is considered full when the next write pointer (wgraynext) matches the synchronized read pointer for the write domain (wq2_rptr). The wfull signal reflects thefull condition.*

***Design Considerations***

*The module consists of several submodules for handling synchronization of read and writepointers, managing the FIFO memory, and handling full and empty conditions. sync_r2w synchronizes the read pointer (rptr) from the read clock domain to the write clock domain. sync_w2r synchronizes the write pointer (wptr) from the write clock domain to the read clock domain. fifomem is the core FIFO memory module that handles write and read operations synchronously. rptr_empty is responsible for generating the rempty signal based on the read pointer and empty condition. wptr_full is responsible for generating the wfull signal based on the write pointer and full condition.*

***Design Considerations***

*The design is modular, separating concerns and responsibilities into distinct submodules.External control of write and read operations is assumed to prevent overflow and underflow conditions. Lack of built-in synchronization for read and write operations acrossdifferent clock domains implies that external synchronization measures should be implemented. The design is optimized for ease of integration and minimal resource utilization.*

### 7.2.6 FIFO Full

***Ports Description***

*wclk: Write clock signal (input). The rising edge triggers write operations into the FIFO memory.*
*wrst_n: Write domain reset, active low (input). The reset signal for the write domain. Whenasserted low, it asynchronously resets the output pointers to zero.*
*wfull: Write full signal (input). When high, it indicates the FIFO memory is full, and no further writeoperations should be attempted.*

*wptr: Write pointer (input). A binary vector of width ADDRSIZE + 1 bits, representing thecurrent write address in the write domain.*

## 7.3 <SystemVerilog abstraction Features used>

***Interface**: Known as an interface bus, it offers a clear, hierarchical design structure while encapsulating a group of signals. **always_ff**: This expression follows best coding principles for synchronous architecture and describes sequential logic (clocked always block).*
***always_comb**: These keywords are used to describe combinational and sequential logicblocks, respectively, making the code easier to read and manage. **parameters**:Parameters are used to provide configurability to the module. **localparam:** The localparam keyword is used to define local parameters within the modules. **logic:** 4 state logic data type used for signal declarations can be used both in procedural and continuous assignments.*

## 7.4 <Simulation, Tools, Directory Structure>

*Questasim*

# 8. Verification

## 8.1 Testing Strategies

**Testing Methods:**

*Black-box testing will be employed to validate the functional correctness of each module.White-box testing will be used to ensure comprehensive coverage, focusing on internal logic and code paths.*
*Gray-box testing will be utilized to combine elements of both black-box and white-boxtesting.*

***PROs and CONs**:*

*Black-box testing PROs: Emphasizes functional correctness, abstraction of internal details.*
*Black-box testing CONs: Limited insight into internal logic and coverage.*
*White-box testing PROs: Comprehensive coverage, thorough understanding of internallogic.*
*White-box testing CONs: May miss high-level functional issues.*

***Testbench Architecture:***

*A modular testbench architecture will be employed with separate testbenches for eachmodule. Common components include drivers, monitors, scoreboards, and checkers for effective testbench functionality.*
*Verification Strategy:*

*Dynamic simulation will be the primary verification strategy.*
*Formal verification may be considered for critical modules or scenarios.*
Emulation will be explored if required for performance validation.

## 8.2 Test case scenarios

***Testing Methodology:***

*A mix of directed and constrained-random testing will be employed.*
*Constrained-random testing will focus on exploring corner cases and potential boundary conditions.*

***Driving Methodology:***

*Test generation methods will include a combination of directed testing for specific scenarios and constrained-random testing for broader coverage.*

***Checking Methodology:***

*Checking will involve comparing expected outputs with simulated results.*
*Assertions and coverage metrics will be used to ensure the design meets specified criteria.*
(detail Testbench plan is given in the Verification plan document)

| ***wdata*** | ***wrst*** | ***rrst*** | ***Description*** |
|---|---|---|---|
| *44* | *0* | *0* | *Basic write operation with both wrst and rrst as 0.* |
| *45* | *1* | *1* | *Basic writes with continous numbers.* |
| *46* | *1* | *1* | *Basic writes with continous numbers.* |
| *47* | *1* | *1* | *Basic writes with continous numbers.* |
| *48* | *1* | *1* | *Basic writes with continous numbers.* |
| *49* | *1* | *1* | *Basic writes with continous numbers.* |

| | | | |
|---|---|---|---|
| *FF* | *1* | *1* | *Edge Cases:Wdata check for ff and 00 - Generate transactions with border values (0 and 255) for wdata. Ensure that both bins for wdata_border are hit.* |
| *0* | *1* | *1* | *Edge Cases:Wdata check for ff and 00 - Generate transactions with border values (0 and 255) for wdata. Ensure that both bins for wdata_border are hit.* |
| *1* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *2* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *4* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *8* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *10* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *20* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *40* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *80* | *1* | *1* | *Wdata check one hot bits - Generate transactions with different one-hot encoded values for wdata.* |
| *1* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *2* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |

| | | | |
|---|---|---|---|
| *3* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *4* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *5* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *6* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *7* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *8* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *9* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *10* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *11* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *12* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *13* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *14* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *15* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |

| | | | |
|---|---|---|---|
| *16* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *17* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *18* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *19* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *20* | *1* | *1* | *Test with specific patterns that may exercise specific functionality or corner cases in your design.* |
| *zz* | *1* | *1* | *Passing unknown values and verifying assertions.* |
| *x* | *1* | *1* | *Passing unknown values and verifying assertions.* |

### 8.3 Others

# Summary

*The verification plan outlines the strategy, methodologies, and specific test cases to ensure the thorough verification of the FIFO design. The combination of directed and random testing, along with appropriate checking mechanisms, aims to uncover potentialissues and ensure a robust and reliable design. The plan will be continuously updated based on feedback and evolving project requirements.*