High Level Design Specification (HLDS)

for

Asynchronous FIFO Design and Verification

Version 1.0

Prepared by Team - 12

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

01/26/2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of this High Level Design Specification (HLDS) document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at high level. This document covers the design specifications and verification of an Asynchronous FIFO with multiple clock domain frequencies. This document is the version 1.0 and can be subjected to changes if needed in future. The HLDS uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

1.2 Document Conventions

1.2.1 Syntax Conventions

Convention	Description
Bold (Times new Roman - 18)	*Section Main Headings*
Bold (Times new Roman - 14)	*Section Subheadings inside Main Headings*
Italic (Arial - 11)	*Description having information about the headings, that consists of theoretical explanation or numerical expressions/values*
Italic, Bold (Times new Roman - 11)	*Headers and Footers of a page has information about page numbers, course number, course title and the term*

1.2.2 Naming Conventions

Convention	Description
FIFO	*First-in-First-out*
HLDS	*High level design specification*
DWIDTH	*Data Width*
ADDRWIDTH	*Address Width*
Posedge	*Positive edge of Clock*
wclk, rclk	*Producer and Consumer Clocks*

wrst_n, rrst_n	*Producer and consumer active low resets*
w_enable, r_enable	*Write enable, Read enable*
wptr, rptr	*Write and Read Pointers*
wb_ptr, rb_ptr	*Binary Write and Read Pointers*
wg_ptr, rg_ptr	*Gray Write and Read Pointers*
wgptr_sync, rgptr_sync	*Synchronized Gray Write and Read Pointers*
waddr, raddr	*Write Address, Read Address*
wdata, rdata	*Write data, Read data*

1.3 Intended Audience and Reading Suggestions

1.3.1 Design Engineers

Professionals and students specializing in digital design, especially those working on memory systems, data storage, and digital circuits, would benefit from understanding the principles and intricacies of asynchronous FIFOs.

1.3.2 Hardware Engineers

Hardware engineers involved in digital design and FPGA/ASIC development would find the document valuable. It could cover details about the design, implementation, and considerations specific to hardware aspects

1.3.3 Verification Engineers

Engineers involved in verification and testing of digital designs, especially those focusing on correctness, timing, and reliability aspects, may find information in the document relevant to their work.

1.3.4 Researchers and Academia

Researchers and academics in the field of digital design, computer engineering, or related areas may use the document as a reference for teaching, research, or further exploration of asynchronous FIFO concepts.

1.3.5 Students in Digital Design Courses

Students studying digital design, computer architecture, or related courses can benefit from a document that provides an introduction to asynchronous FIFOs, their principles, and considerations in design.

1.3.6 System Architects

System architects who are responsible for designing complex systems that involve data storage, communication between components, or synchronization across clock domains could benefit from understanding the role and characteristics of asynchronous FIFOs.

1.4 Product Scope

Using an asynchronous FIFO (First-In-First-Out) with multiple clock domain frequencies offers several benefits in digital design, especially when dealing with components that operate at different clock rates. Some advantages are Cross-clock domain data transfer, decoupling clock domains, handling clock skew, flexibility in system integration, mitigating metastability, avoiding clock domain frequency matching, reducing system complexity, and scalability.

1.4.1 Crossing Clock Domains

Async FIFOs are crucial when transferring data across clock domains within a larger System-on-Chip (SoC) or FPGA design. They ensure proper synchronization and prevent data loss or corruption.

1.4.2 Decoupling Clock Domains

Asynchronous FIFOs decouple the write and read operations, allowing components in different clock domains to operate independently. This decoupling enhances modularity and simplifies the overall system design.

1.4.3 Mitigating Metastability

Asynchronous FIFOs include techniques such as dual flip-flop synchronizers to mitigate metastability issues that can arise when data transitions between clock domains. This ensures the robustness of data transfer even in the presence of clock domain crossings

1.4.4 Avoiding Clock Domain Frequency Matching

Synchronous FIFOs would require the clock domains to have matching frequencies, which might not always be feasible or practical. Asynchronous FIFOs eliminate the need for matching clock frequencies, allowing for more diverse system configurations.

1.4.5 Handling Clock Skew

Clock skew, which is the variation in arrival times of clock signals, is more prevalent when dealing with different clock domains. Asynchronous FIFOs are designed to handle such skew, ensuring reliable data transfer across clock domains.

1.4.6 Scalability

Asynchronous FIFOs are scalable and can accommodate various clock frequencies, making them suitable for designs with evolving requirements and varying clock domains.

1.4.7 I/O Interfaces

Async FIFOs are crucial when transferring data across clock domains within a larger System-on-Chip (SoC) or FPGA design. They ensure proper synchronization and prevent data loss or corruption.

1.5 References

1. *Title: Simulation and Synthesis Techniques for Asynchronous FIFO Design.*
   Author: Clifford E. Cummings.
   Date: 2002
   Source: Simulation and Synthesis Techniques for Asynchronous FIFO Design

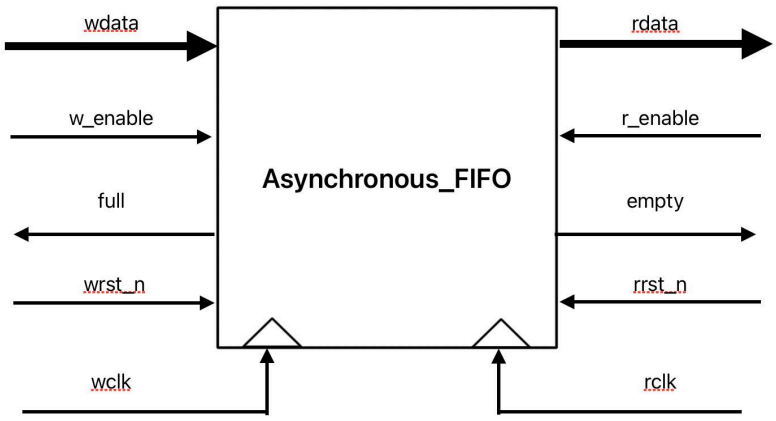
2. *Title: Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons.*
   Author: Clifford E. Cummings, Peter Alfke
   Date: 2002
   Source: Asynchronous FIFO Design with Asynchronous Pointer Comparisons

3. *Title: Crossing clock domains with an Asynchronous FIFO*
   Author: David Harris, Sarah Harris
   Date: 2018
   Source: Crossing clock domains with an Asynchronous FIFO

4. *Title: Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs*
   Author: Clifford E. Cummings
   Date: 2001
   Source: Synthesis Techniques for Designing Multi-Asynchronous Clock Designs

2. Overall Description

2.1 Product Perspective



Asynchronous FIFO is used to transfer data from one module to another module with different clock domain frequencies. The proposed design encourages the clock domain isolation and serves as a buffer between components operating in different clock domains. It isolates the write and read operations, allowing them to proceed independently. This asynchronous FIFO with multiple clock domain frequencies can be used as a small component in designing communication systems, microprocessor systems, image and video processing systems, SoC designs and mixed signal systems.

The asynchronous FIFO proposed in this HLDS have FIFO depth = 228 and FIFO width = 8. The data is typically written at the positive edges (posedge) of the write clock domain. The write clock domain is associated with the clock signal used for writing data into the FIFO, this ensures that the data is stable and can be reliably captured by the receiving clock domain. The interaction between two clock cycles is relatively straightforward. During the write operation, the exact movement of write is determined by the Write Enable signal. The data written is stored in the FIFO memory. The read addressing in the FIFO is controlled by the read clock and it operates independently. FIFO memory should be designed to allow for reliable data storage and retrieval within the same clock cycle for both read and write operations. During the read operation, the exact moment of read is determined by the Read Enable signal. Before the positive edge of the read clock the data to be read should be available and stable in the FIFO memory.

2.2 Product Functions

2.2.1 Primary Features and Capabilities

- ➔ *Data Buffering*
- ➔ *Cross-clock domain data transfer*
- ➔ *Clock domain decoupling*
- ➔ *Metastability handling*
- ➔ *Flexible memory depth*
- ➔ *Synchronization and Alignment*
- ➔ *Scalability*
- ➔ *Adaptable to system changes*

2.2.2 Major groups of the top level design

2.2.2.1 External components

- *Write component: Represents the component responsible for writing data into the Asynchronous FIFO.*
- *Read component: Represents the component responsible for reading data from the Asynchronous FIFO.*
- *Clock domains: Represents different clock domains associated with write and read components.*

2.2.2.2 Asynchronous FIFO

- *Data Buffer: Represents the storage area where data is temporarily stored.*
- *Write pointer: Represents the pointer that indicates the write location within the FIFO.*
- *Read pointer: Represents the pointer that indicates the read location within the FIFO.*
- *Control logic: Manages the control signals like, write enable, read enable.*

2.2.2.3 Control Signals

- *Write Enable: Represents the signal indicating when the data should be written*
- *Read Enable: Represents the signal indicating when the data should be read*
- *Clock Signals: Represents clock signals associated with write and read clock domains*

2.2.2.4 Clock Domain Crossing

- *Synchronization Logic: Represents the synchronization techniques used to handle clock domain crossings and mitigate metastability.*

2.3 User Classes and Characteristics

The user classes of this asynchronous FIFO would be Digital Design engineers, FPGA and ASIC designers, Communication System engineers, Signal Processing engineers, Verification and Validation engineers, R&D teams, Academia, Power Management specialists. These are the broad spectrum of professionals in different domains who may use the FIFO for its diverse applications.

Some of the characteristics of this asynchronous FIFO with multiple clock frequencies that other users could find useful to integrate in their designs,

- ➔ *Metastability Handling*
- ➔ *Clock Domain Independence*
- ➔ *Configurable FIFO Depth*
- ➔ *Configurable FIFO Width*
- ➔ *Power management*
- ➔ *Simple Interface*
- ➔ *Compatibility with standard design tools*
- ➔ *Test benches and verification tools*

2.4 Tools and Software

The software operates in an environment where FPGA development boards from vendors like Xilinx, Intel, or Lattice Semiconductor are used. These boards provide a hardware platform for prototyping, testing, and implementing designs with asynchronous FIFOs.

2.4.1 HDL editors

Hardware description language editors such as Modelsim, Questasim, VCS can be used. These tools are used for writing, editing and simulating a Verilog code.

2.4.2 Synthesis and Implementation tools

Synthesis tools like Xilinx Vivado, Intel Quartus Prime can be used. They convert RTL code into gate level netlists for hardware platforms.

2.4.3 Timing Analysis tools

Static Timing Analysis (STA) tools like Prime time or Tempus can be used to verify timing characteristics of the asynchronous FIFO design.

2.4.4 Operating System and Environment

Commonly used operating systems are Windows, Linux, or MacOS. The s/w operates in any environment that supports chosen design tools and is compatible with OS requirements.

2.5 Design and Implementation Constraints

2.5.1 Design Limitations

2.5.1.1 Metastability Challenges

Asynchronous FIFOs are vulnerable to metastability issues during clock domain crossings. Even with dual flip-flop synchronizers, there is a small but non-zero probability of metastability, this can result in corruption of data.

2.5.1.2 Increased Latency

Implementing asynchronous FIFOs introduces additional complexity compared to synchronous designs. Handling asynchronous events and ensuring correct operation across multiple clock domains requires careful consideration and design effort.

2.5.1.2 Testing and Verification Challenges

Testing and verifying asynchronous FIFO can be more challenging compared to synchronous designs. Special attention is needed to ensure robustness in the face of different clock domain scenarios.

2.5.2 Implementation Constraints

2.5.2.1 *Clock domain crossing constraints*

Implementing robust clock domain crossing (CDC) strategies is crucial. Constraints must be applied to ensure proper synchronization of signals crossing between different clock domains. This includes using dual flip-flop synchronizers, proper handshake protocols, and managing metastability.

2.5.2.2 *Setup and Hold time constraints*

Meeting setup and hold time constraints becomes challenging in asynchronous designs. Designers must carefully analyze the timing requirements and ensure that data is stable and valid for the required setup and hold times.

2.5.2.3 *FIFO Depth constraints*

Determining the appropriate FIFO depth is critical. Constraints related to the depth of the FIFO need to be considered to prevent overflow or underflow conditions. The depth should be sufficient to accommodate variations in data rates between clock domains.

2.5.2.4 *Data Width and word alignment constraints*

Constraints related to the data width of the FIFO and word alignment must be defined. The data width should match the requirements of the connected components, and proper alignment of data words should be maintained to avoid data corruption.

2.5.2.5 *Maximum Operating Frequency constraints*

Asynchronous FIFOs may have limitations on the maximum operating frequencies due to the complexity of asynchronous design techniques. Designers must ensure that the chosen clock frequencies allow for reliable and stable operation.

2.6 Assumptions and Dependencies

2.6.1 Independent Clock Domains

The design assumes that the write and read clock domains are independent and have different frequencies. Synchronization techniques, such as dual flip-flop synchronizers, are applied to handle clock domain crossings.

2.6.2 Metastability Handling

It is assumed that metastability issues during clock domain crossings will be addressed using appropriate synchronizers. The assumption is that dual flip-flop synchronizers or similar techniques are effective in minimizing the risk of metastability.

2.6.3 Dependency on correct FIFO Depth

The assumption is made that designers will select an appropriate FIFO depth based on the anticipated variations in data rates between the write and read clock domains. A proper FIFO depth is critical to prevent overflow or underflow conditions.

2.6.4 Proper Data Alignment

The design assumes that data alignment within the FIFO is maintained correctly. This includes ensuring that data words are aligned according to the requirements of the connected components in both clock domains.

2.6.5 Dependency on Tool Compatibility

The design assumes compatibility with the synthesis, place-and-route, and simulation tools used. Dependencies exist on the correct interpretation of constraints and design properties by these tools.

3. External Interface Requirements

3.1 Hardware Interfaces

The hardware interface of an asynchronous FIFO contains different hardware interfaces among which few critical aspects have been mentioned below.

3.1.1 Asynchronous Data I/O: *Data inputs and outputs operate independently, potentially across different clock domains, necessitating synchronization mechanisms to ensure reliable communication.*

3.1.2 Control Signals: *Manage asynchronous data transfers, including handshaking signals for data readiness and flow control.*

3.1.3 Clock Domain Crossing: *Employ techniques such as synchronizers to handle data transfers between asynchronous clock domains, mitigating timing issues and ensuring data integrity.*

3.1.4 Pointer Management: *Coordinate write and read pointers across clock domains, employing safe updating methods to prevent data corruption or loss.*

3.1.5 Flexible Data Handling: *Support various data widths and depths to accommodate diverse processing requirements.*

3.1.6 Initialization and Reset: *Initialize and reset internal states while considering asynchronous clock domains, ensuring proper operation and synchronization.*

3.1.7 Error Handling: *Implement robust error detection and recovery mechanisms to manage overflow, underflow, and data corruption issues across asynchronous domains.*

This comprehensive hardware interface facilitates seamless and reliable data transfer in asynchronous environments, ensuring the FIFO's effectiveness in diverse digital systems.

3.2 Software Interfaces

The software interfaces of an asynchronous FIFO with multiple clock domain frequencies typically refer to the ways in which software or higher-level components interact with the FIFO at the software level.

3.2.1 Application Programming Interface (API)

An API provides a set of functions or methods that software applications use to interact with the asynchronous FIFO. This can include functions for writing data to the FIFO, reading data from the FIFO, checking status flags, and managing reset operations.

3.2.2 Register Based Interface

The FIFO may expose a set of registers that software can read from or write to. These registers control various aspects of FIFO operation, such as configuration settings, status flags, and pointers.

3.2.3 Interrupts and Status Flags

The FIFO may generate interrupts or expose status flags that software can monitor to be informed about events such as FIFO full, FIFO empty, or other conditions affecting data transfer.

3.2.4 Synchronization Mechanisms

In some cases, software interfaces may include synchronization mechanisms to coordinate data transfer between different clock domains. This could involve setting up handshaking protocols or managing flow control.

3.2.5 FIFO Reset and Initialization

Commands or functions for resetting and initializing the asynchronous FIFO may be part of the software interface. This includes options for synchronous or asynchronous resets and setting the initial state of the FIFO.

4. Product Features

4.1 FIFO Memory

- ***Storage and Read Addressing:*** *The data written at the positive edge of the write clock is stored in the FIFO memory. The read addressing in the FIFO, controlled by the read clock, operates independently*
- ***Memory Access Time:*** *The FIFO memory should be designed to allow for reliable data storage and retrieval within the same clock cycle for both read and write operations.*

4.2 Synchronization

- *While not strictly necessary, some designers might choose to include minimal synchronizers in the design for added robustness against potential variations in clock signals, noise, or other factors.*

4.3 Gray Code Counters

- *Gray code counters are binary counters where only one bit changes at a time, reducing the chance of errors in case of metastability.*
- *Gray code can be advantageous in asynchronous designs where transitions between states should minimize the risk of metastability.*

4.4 Write Operation

- ***Write Clock Edge:*** *The data is typically written into the FIFO on the positive edge of the write clock. The exact moment of the write is determined by the Write Enable signal (WE).*
- ***Data Stability****: Before the positive edge of the write clock, the data to be written should be stable and valid.*

4.5 Read Operation

- ***Read Clock Edge:*** *The data is typically read from the FIFO on the positive edge of the read clock. The exact moment of the read is determined by the Read Enable signal.*
- ***Data Availability****: Before the positive edge of the read clock, the data to be read should be available and stable in the FIFO memory.*

4.6 Configurable Depth

- *Allows users to configure the depth of the FIFO to match specific application requirements, preventing data loss or overflow.*

5. Logic Design

5.1 Work Directory Structure

As part of our design code maintenance and integration we have implemented usage of GIT which contains details about our Asynchronous FIFO modules and testbenches.

***Git**: Github Repository Link*

More about design functionality can be found in the section 5.2 below, which provides detailed information about what each design module contains in terms of signals and logic to a certain extent. Users can try running the design by cloning the design modules and test benches from GIT and can run the Asynchronous FIFO by using run.do file which is shared in the same repo.
FYI Info regarding verification plan and future activities are updated in below document for tracking purpose.

***Verification Plan:** Verification Plan*

5.2 Design modules

To facilitate static timing analysis of the asynchronous FIFO design, the design has been partitioned into the following seven Verilog modules with the following functionality and clock domains:

- ***Asynchronous_FIFO.sv:** This is the top-level wrapper-module that includes all clock domains. The top module is only used as a wrapper to instantiate all of the other FIFO modules used in the design. If this FIFO is used as part of a larger ASIC or FPGA design, this top-level wrapper would probably be discarded to permit grouping of the other FIFO modules into their respective clock domains for improved synthesis and static timing analysis.*

- ***FIFO_Memory.sv**: This is the FIFO memory buffer that is accessed by both the write and read clock domains. This buffer is most likely an instantiated, synchronous dual-port RAM. Other memory styles can be adapted to function as the FIFO buffer.*

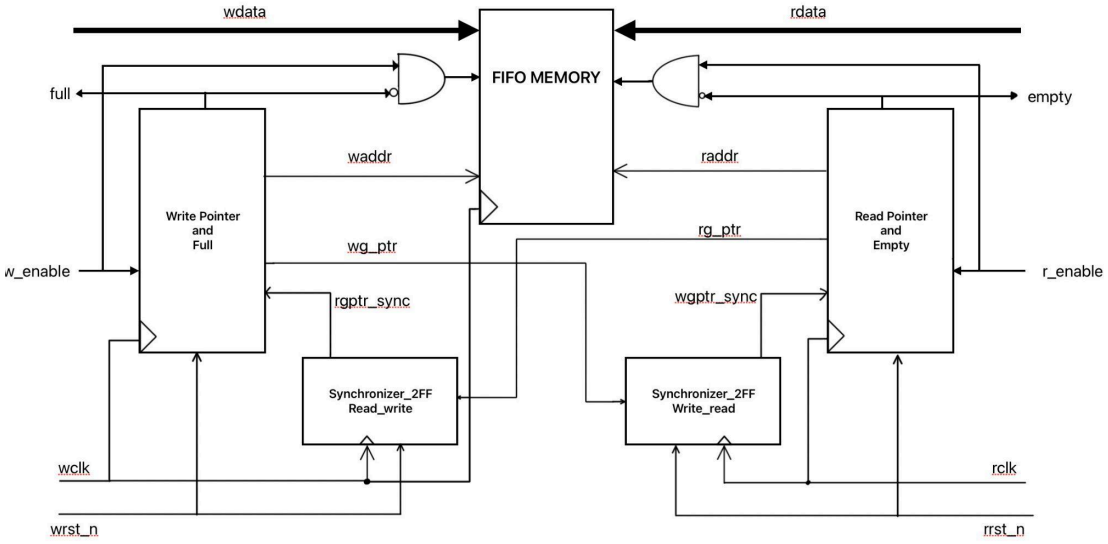
- ***Read_Pointer_Empty.sv:** This module is mostly synchronous to the read-clock domain and contains the FIFO read pointer and empty-flag logic.*

- ***Write_Pointer_Full.sv:** This module is mostly synchronous to the write-clock domain and contains the FIFO write pointer and full-flag logic.*

- ***Gray_to_Binary_converter:** Gray-to-binary conversion logic in the asynchronous FIFO is used to translate incoming gray code into binary, which is crucial for synchronizing data between asynchronous clock domains. It decodes gray code using combinatorial logic, ensuring accurate data transfer across domains. This conversion facilitates communication between components operating at different clock frequencies.*

- ***Binary_to_gray_converter:** Binary-to-gray conversion logic in the asynchronous FIFO is used to translate incoming binary code into gray, which is crucial for synchronizing data between asynchronous clock domains. It decodes binary code using combinatorial logic, ensuring accurate data transfer across domains. This conversion facilitates communication between components operating at different clock frequencies.*

- ***synchronizer.sv**: Synchronization and transmitting of signals is done in this module i.e., passing the input signals into 2FF and gives the output signals to the respective modules.*



5.3 SystemVerilog abstraction Features

5.3.1 FIFO Parameterization:

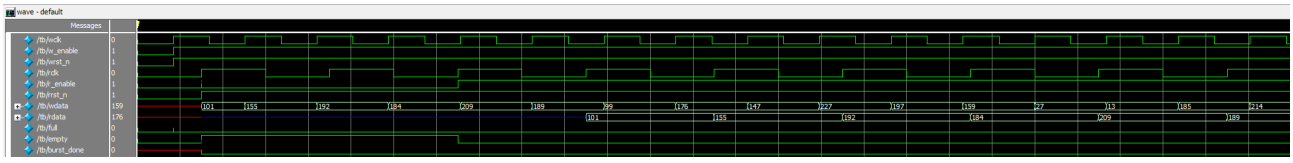
DWIDTH *- This parameter represents the number of bits per entry in the FIFO.*
ADDRWIDTH *- This parameter is automatically filled by the $\log_2$ macro to be the number of bits for your read and write pointers.*

5.3.2 Datatypes

All signals, inputs and Outputs are declared as Logic variables. The reason to choose all of them as logic is it will be useful to identify bugs with ease while debugging errors if there are any.

5.4 Simulation, Tools, Directory Structure

Simulation is conducted using QuestaSim version 10.7, a platform recognized for its robust support of SV and UVM methodologies. The operating system for our development environment is Microsoft Windows 11. This combination ensures a streamlined integration with Questa Sim, enhancing the efficiency of our verification tasks. The use of SV and UVM facilitates a systematic and standardized approach to validating the Async FIFO design.



6. Verification

As a part of verification we have planned on verifying the design using below methods:

- ➔ *Assertion based Verification*
- ➔ *Functional verification*
- ➔ *Class based verification*
- ➔ *Environment implementation(monitors,score boards, Checkers, Drivers,Interfaces)*
- ➔ *UVM Based verification etc….*

6.1 TestBench Style

We have planned on using a structured approach to ensure thorough verification of digital designs. The test bench architecture incorporates reusable verification components such as UVCs (Universal Verification Components), sequences, and drivers to generate stimulus. Assertions embedded within the test bench enforce design properties and constraints, enhancing the detection of errors and violations. Scoreboards, checkers, and monitors analyze DUT (Design Under Test) outputs against expected results, providing real-time feedback on design correctness. This combination of UVM methodology, assertions, scoreboards, checkers, and monitors fosters comprehensive verification, improving confidence in design functionality and accelerating the verification process in complex semiconductor projects.

6.2 *Test case scenarios*

Below test case scenarios has been included and have been planned to be implemented during the verification planning and applying design, verification methodologies in different scenarios:

Input Validation:
- *Test cases to validate input data against expected formats and ranges.*
- *Verify handling of invalid or out-of-range inputs.*

Boundary Conditions:
- *Evaluate behavior at boundary conditions, such as minimum and maximum values.*
- *Test upper and lower limits of data ranges.*

Error Handling:
- *Validate the response to error conditions, such as buffer overflows, timeouts, or hardware faults.*
- *Verify error messages, alarms, or corrective actions.*

Regression Testing:
- *Re-run previously passed test cases to ensure that recent changes have not introduced new defects.*
- *Validate that fixes for reported issues do not cause regression in other areas*

6.3 Testing Strategies

As part of test strategy we have below strategies to implement different aspects as mentioned below
Requirement Analysis: Begin by understanding design specifications and defining verification goals and coverage metrics.
- *Test Plan Development: Create a comprehensive test plan outlining test scenarios, coverage goals, and verification methodologies such as UVM.*
- *Test Environment Setup: Establish a well-structured test bench environment using UVM methodology, including reusable verification components, sequences, and drivers.*

- *Stimulus Generation: Develop stimulus generation techniques, including constrained randomization, to generate diverse and realistic test scenarios.*
- *Assertion Integration: Embed assertions within the test bench to formalize design properties, constraints, and expected behaviors.*
- *Coverage Closure: Monitor and analyze functional coverage metrics to ensure comprehensive verification and identify coverage gaps.*
- *Debugging and Error Analysis: Implement debug features, transaction-level debuggers, and visualization tools to facilitate efficient error detection and diagnosis.*

- *Continuous Improvement: Iterate on test cases, enhance verification environments, and refine test strategies based on feedback and evolving requirements.*
  By following these strategies we have planned on thorough validation, early bug detection, and accelerated time-to-market for digital designs.

6.4 Others

In addition to implementing test plans and following strategies there should also be track of all the actions by documenting what has been planned to be done as part of information sharing, tracking for other resources. Defining a structured approach for implementing test cases, including scheduling, resource allocation, and assignment of responsibilities. Establishing clear milestones and deliverables to track progress throughout the verification process. Even though there is a planned approach the design and verification process are subject to be improvised as per the requirements.

Summary

Appendix A: Glossary

Syntax Conventions

Convention	Description
Bold (Times new Roman - 18)	*Section Main Headings*
Bold (Times new Roman - 14)	*Section Subheadings inside Main Headings*
Italic (Arial - 11)	*Description having information about the headings, that consists of theoretical explanation or numerical expressions/values*
Italic, Bold (Times new Roman - 11)	*Headers and Footers of a page has information about page numbers, course number, course title and the term*

Naming Conventions

Convention	Description

FIFO	*First-in-First-out*
HLDS	*High level design specification*
DWIDTH	*Data Width*
ADDRWIDTH	*Address Width*
Posedge	*Positive edge of Clock*
wclk, rclk	*Producer and Consumer Clocks*
wrst_n, rrst_n	*Producer and consumer active low resets*
w_enable, r_enable	*Write enable, Read enable*
wptr, rptr	*Write and Read Pointers*
wb_ptr, rb_ptr	*Binary Write and Read Pointers*
wg_ptr, rg_ptr	*Gray Write and Read Pointers*
wgptr_sync, rgptr_sync	*Synchronized Gray Write and Read Pointers*
waddr, raddr	*Write Address, Read Address*
wdata, rdata	*Write data, Read data*