

# **DESIGN AND VERIFICATION OF A POINT-TO-POINT WISHBONE INTERCONNECT**

**A MINI PROJECT REPORT**

*Submitted by*

**Dhinesh Kumar S (2022105012)**

**Shailendher A (2022105014)**

**Sabarinath N (2022105033)**

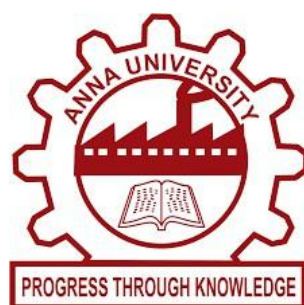
*In partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**



**COLLEGE OF ENGINEERING GUINDY,**

**ANNA UNIVERSITY: CHENNAI 600 025**

**NOVEMBER 2025**

## **BONAFIDE CERTIFICATE**

Certified that this project report “**DESIGN AND VERIFICATION OF A POINT TO POINT WISHBONE INTERCONNECT**” is the Bonafide work of “**Dhinesh Kumar S** (2022105012), **Shailendher A** (20221055014), **Sabarinath N** (20221055033)” who carried out the project work under my supervision.

**SIGNATURE**

**Dr. M. A. BHAGYAVENI**

**HEAD OF THE DEPARTMENT**

**PROFESSOR**

Department of Electronics and  
Communication Engineering  
College of Engineering Guindy,  
Anna University,  
Chennai – 600 025

**SIGNATURE**

**Dr. S. R. SRIRAM**

**SUPERVISOR**

**ASST PROFESSOR**

Department of Electronics and  
Communication Engineering  
College of Engineering Guindy,  
Anna University,  
Chennai – 600 025

## **ACKNOWLEDGEMENT**

We express our wholehearted gratitude to our dynamic and zestful Head of the Department Dr. M. A. BHAGYAVENI, Professor, Department of Electronics & Communication Engineering for her continued and heartening support throughout the project.

We manifest our sincere thanks and regards to our project supervisor, Dr. S. R. SRIRAM, Professor, Department of Electronics and Communication Engineering for her ardent support, patience, valuable guidance, technical expertise and encouragement in our project.

We also are supremely grateful to the teaching and non-teaching staff of the Department of Electronics and Communication Engineering, for their kind help and co-operation during the course of our project.

Last but not the least, we would not have made this project without the support from our beloved family and friends.

## ABSTRACT

The Wishbone bus is an open-source on-chip interconnection architecture that standardizes communication between intellectual property (IP) cores in system on-chip (SoC) designs. In this work, the design and verification of a point-to point Wishbone bus interface are carried out using SystemVerilog. The design emphasizes the correct implementation of core bus signals and protocols, including cycle initiation, data transfer, read/write control, and acknowledgment mechanisms. Particular focus is given to supporting both single read/write transactions and block read/write transfers, ensuring efficient data movement between master and slave devices. The bus interface is developed for multiple bus widths (8, 16, 32, and 64 bits) to demonstrate scalability. To validate the functionality, SystemVerilog testbenches are created to verify protocol compliance, timing correctness, and data integrity under different scenarios, including wait-state insertion during block transfers. By leveraging SystemVerilog's advanced verification features, the Wishbone point-to-point interface is thoroughly validated, ensuring robustness in handling both single and burst transactions. This work shows that Wishbone, with systematic verification, provides a lightweight yet reliable interconnection framework for modern SoC designs.

## TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	3
1	INTRODUCTION	5
2	BLOCK DIAGRAM	6
3	LITERATURE REVIEW	6
4	PROPOSED WORK	7
5	WISHBONE CODE	9
	DESIGN CODE	9
	TESTBENCH CODE	17
6	OUTPUT	30
7	REFERENCES	42

# CHAPTER 1

## INTRODUCTION

The Wishbone Bus Protocol is an open-source standard designed to connect various Intellectual Property (IP) cores within a System-on-Chip (SoC). It defines a flexible and uniform communication interface that allows different functional modules—such as processors, memory units, and peripheral devices—to interact seamlessly on a single chip. By providing a common communication framework, Wishbone simplifies SoC integration, improves design reusability, and reduces overall development time and complexity while maintaining high performance and scalability.

Wishbone supports four main interconnection topologies:

- **Point-to-Point:** A single master is directly connected to a single slave. It is simple, fast, and requires no arbitration. (This configuration is used in our miniproject.)
- **Shared Bus:** Multiple masters share a common bus and communicate with multiple slaves through an arbiter, which grants access one at a time.
- **Crossbar:** Allows multiple masters to communicate with multiple slaves simultaneously using independent paths, improving speed but increasing hardware complexity.
- **Dataflow:** Modules are connected in a pipeline fashion, where data flows sequentially from one block to another, ideal for streaming or signal-processing applications.

The Wishbone Master uses signals — RST\_I, CLK\_I, ADR\_O, DAT\_I, DAT\_O, WE\_O, SEL\_O, STB\_O, ACK\_I, CYC\_O, TAGN\_I, TAGN\_O, and CTI\_O[2:0].

The Wishbone Slave uses signals — RST\_I, CLK\_I, ADR\_I, DAT\_I, DAT\_O, WE\_I, SEL\_I, STB\_I, ACK\_O, and CYC\_I.

This miniproject implements a Point-to-Point Wishbone interface where a single master communicates directly with a single slave to perform classic and block read/write operations using the CTI-based cycle control mechanism.

## CHAPTER 2

### BLOCK DIAGRAM

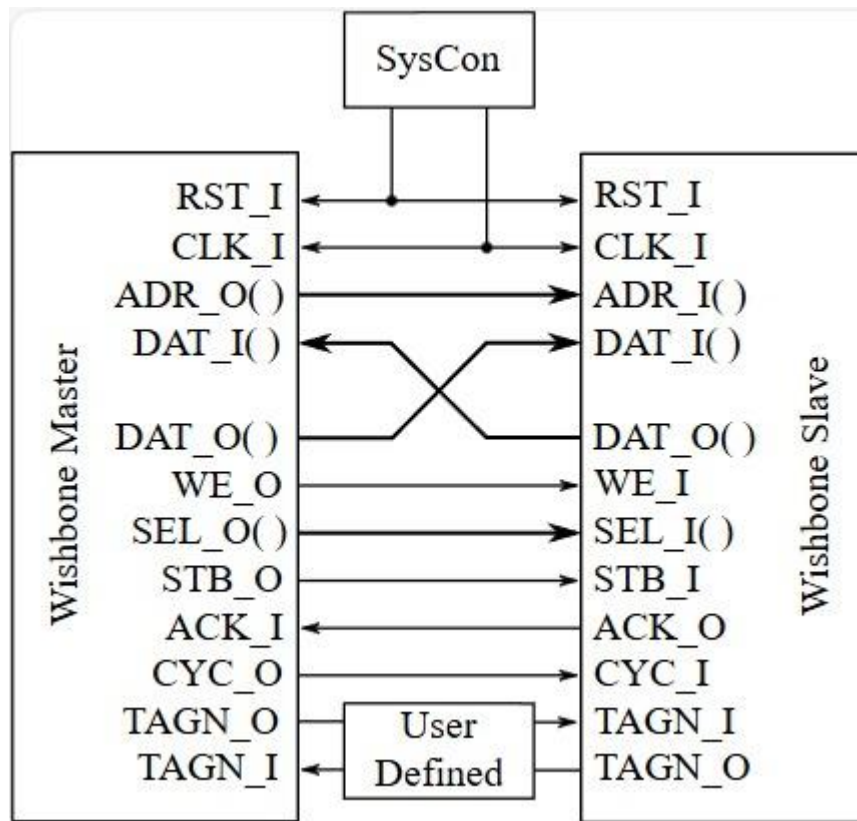


Fig 1: Point to point Architecture

## CHAPTER 3

### LITERATURE REVIEW

- Earlier works focused on designing Wishbone bus interfaces for smooth data transfer. They implemented master and slave modules for reliable communication. Simulation proved correct timing and protocol operation.
- Previous studies developed dataflow-based Wishbone architectures for faster processing. They achieved high frequency and low hardware utilization using FPGA implementation. This improved speed and reduced delay in SoC communication.

- Comparative analyses were done between Wishbone and other bus standards.

It was found that Wishbone is flexible, open-source, and easy to integrate. Thus, it is widely used for low-cost and efficient SoC designs.

- The WISHBONE B4 specification defines a simple, synchronous bus interface for SoC integration. It supports classic, block, and read-modify-write cycles for flexible data transfer. B4 improves system performance, portability, and IP core reusability in FPGA and ASIC designs.

## **CHAPTER 4**

### **PROPOSED WORK**

The proposed work focuses on implementing and verifying a Wishbone bus system supporting both classic single-cycle and block read/write transfers using CTI signals. The design includes a Wishbone master, slave, and top-level interconnect, where the master controls data flow and the slave performs memory operations with ACK and ERR responses. The slave also supports a special tag-add feature that returns the sum of two memory locations. A System Verilog testbench with generator, driver, monitor, and scoreboard is developed to verify functionality under various CTI conditions (000, 001, 010, 111). The scoreboard maintains a reference memory model to validate data integrity and error handling. The system is tested for valid, invalid, and burst transactions to ensure protocol correctness. Overall, the Mini project demonstrates successful implementation of Wishbone protocol operations, accurate block transfer handling, and reliable verification through simulation.. The proposed work is divided into the following phases:

#### **1. Design of Wishbone Master**

Implement a finite state machine (FSM) consisting of IDLE, BUS\_REQUEST, and BUS\_WAIT states. Generate Wishbone control signals: cyc\_o, stb\_o, we\_o, sel\_o, cti\_o, and tag\_add. Handle single, block, and tag-based read/write operations. Capture received slave data, error, and acknowledge signals.

#### **2. Design of Wishbone Slave**

Implement a synchronous memory model with separate read, write, burst, and tag-add functionalities.

Support Wishbone CTI (Cycle Type Identifier) modes:

000 → Classic Single Transfer

001 → Incrementing Burst

010 → Linear Burst with counter

111 → Burst End

Generate appropriate ACK and ERR signals.

Maintain a burst counter for CTI=010 mode.

### 3. Development of Top-Level Wishbone System

Integrate master and slave into a wishbone\_top module. Provide debug visibility of internal signals (dbg\_\* outputs). Ensure proper timing, handshake, and CTI-based burst coordination.

### 4. Development of Testbench Interface

Create wishbone\_if interface for connecting DUT to verification components. Bundle all bus signals and allow virtual interface access.

### 5. Constrained-Random Stimulus Generation

Implement a transaction class with randomizable fields:

data, cti, we,

Apply constraints to generate:

Write cycles

Read cycles

TAG Read cycles

Block read/write bursts (cti=001, 010)

Block end (cti=111)

### 6. Driver Implementation

Convert randomized transactions into actual Wishbone bus signal activity. Drive handshake operations, handle CTI burst sequences, and wait for ACK/ERR. Dump activity for debug.

### 7. Monitor Implementation

Observe bus activity passively using the interface. Reconstruct transactions based on DUT response. Forward reconstructed transactions to scoreboard.

## 8. Scoreboard and Self-Checking Logic

Maintain expected memory (exp\_mem) model.

Verify:

Write → memory update

Read → compare with expected

TAG read → check  $\text{sum}(\text{mem}[0] + \text{mem}[1])$

Burst mode reads/writes

Error handling for invalid address

Display PASS/FAIL messages.

## 9. Test Environment Integration

Combine generator, driver, monitor, and scoreboard into an environment class. Use mailboxes and events for synchronization. Ensure all transactions from generator, driver, and scoreboard match.

## 10. Simulation, Waveform Debug & Results Analysis

Run using VCD waveform dumping.

Capture: CTI sequences, Burst operations, ACK timing. Tag-add computations, Validate protocol correctness and transaction coverage.

# CHAPTER 5 WISHBONE CODE

## 5.1 DESIGN CODE:

```
module wishbone_top #(
    parameter int ADDR_WIDTH = 5,
    parameter int DATA_WIDTH = 32,
    parameter int SEL_WIDTH = DATA_WIDTH/8
)(
    input logic clk_i,
    input logic rst_i,
    input logic [ADDR_WIDTH-1:0] addr_o,
    input logic [DATA_WIDTH-1:0] data_o,
```

```

input logic we_o,
input logic stb_o,
input logic cyc_o,
input logic [SEL_WIDTH-1:0] sel_o,
input logic [2:0] cti_input,
input logic tag_add,
output logic ack_i,
output logic err_i,
output logic [DATA_WIDTH-1:0] data_i,
output logic [1:0] state_out,
output logic [ADDR_WIDTH-1:0] dbg_w_addr,
output logic [DATA_WIDTH-1:0] dbg_w_data_m2s,
output logic [DATA_WIDTH-1:0] dbg_w_data_s2m,
output logic [ADDR_WIDTH-1:0] counter,
output logic dbg_w_we,
output logic dbg_tag_add,
output logic [SEL_WIDTH-1:0] dbg_w_sel,
output logic dbg_w_stb,
output logic dbg_w_cyc,
output logic dbg_w_ack,
output logic dbg_w_err,
output logic [2:0] dbg_cti
);
logic [ADDR_WIDTH-1:0] w_addr;
logic [DATA_WIDTH-1:0] w_data_m2s;
logic [DATA_WIDTH-1:0] w_data_s2m;
logic w_we;
logic [SEL_WIDTH-1:0] w_sel;
logic w_stb;
logic w_cyc;
logic tag_addd;
logic w_ack;
logic w_err;
logic [2:0] cti;
assign dbg_w_addr = w_addr;
assign dbg_w_data_m2s = w_data_m2s;
assign dbg_w_data_s2m = w_data_s2m;
assign dbg_w_we = w_we;
assign dbg_w_sel = w_sel;
assign dbg_w_stb = w_stb;
assign dbg_w_cyc = w_cyc;
assign dbg_w_ack = w_ack;
assign dbg_cti = cti;
assign dbg_w_err = w_err;
assign dbg_tag_add = tag_addd;

wishbone_master #(
    .ADDR_WIDTH(ADDR_WIDTH),
    .DATA_WIDTH(DATA_WIDTH),
    .SEL_WIDTH(SEL_WIDTH)

```

```

) m1 (
  .clk_i(clk_i),
  .rst_i(rst_i),
  .addr_input(addr_o),
  .addr_o(w_addr),
  .data_input(data_o),
  .data_o(w_data_m2s),
  .write_enable(we_o),
  .we_o(w_we),
  .sel_input(sel_o),
  .sel_o(w_sel),
  .strobe(stb_o),
  .stb_o(w_stb),
  .cyc_input(cyc_o),
  .cyc_o(w_cyc),
  .cti_input(cti_input),
  .cti_o(cti),
  .tag_add_i(tag_add),
  .tag_add_out2s(tag_addd),
  .data_i(w_data_s2m),
  .data_received(data_i),
  .ack_i(w_ack),
  .acknowledgement(ack_i),
  .err_i(w_err),
  .error(err_i),
  .state_out(state_out)
);

wishbone_slave #(
  .ADDR_WIDTH(ADDR_WIDTH),
  .DATA_WIDTH(DATA_WIDTH),
  .SEL_WIDTH(SEL_WIDTH)
) s1 (
  .clk_i(clk_i),
  .rst_i(rst_i),
  .addr_i(w_addr),
  .data_i(w_data_m2s),
  .we_i(w_we),
  .sel_i(w_sel),
  .stb_i(w_stb),
  .cyc_i(w_cyc),
  .cti_i(cti),
  .tag_add_i(tag_addd),
  .ack_o(w_ack),
  .err_o(w_err),
  .data_o(w_data_s2m),
  .counter_out(counter)
);

endmodule

```

```

module wishbone_master #(
    parameter int ADDR_WIDTH = 32,
    parameter int DATA_WIDTH = 32,
    parameter int SEL_WIDTH = DATA_WIDTH/8
)(
    input logic clk_i,
    input logic rst_i,
    input logic [ADDR_WIDTH-1:0] addr_input,
    output logic [ADDR_WIDTH-1:0] addr_o,
    input logic [DATA_WIDTH-1:0] data_input,
    output logic [DATA_WIDTH-1:0] data_o,
    input logic write_enable,
    output logic we_o,
    input logic [SEL_WIDTH-1:0] sel_input,
    output logic [SEL_WIDTH-1:0] sel_o,
    input logic strobe,
    output logic stb_o,
    input logic cyc_input,
    output logic cyc_o,
    input logic [2:0] cti_input,
    output logic [2:0] cti_o,
    input logic tag_add_i,
    output logic tag_add_out2s,
    input logic [DATA_WIDTH-1:0] data_i,
    output logic [DATA_WIDTH-1:0] data_received,
    input logic ack_i,
    output logic acknowledgement,
    input logic err_i,
    output logic error,
    output logic [1:0] state_out );

typedef enum logic [1:0] {
    IDLE,
    BUS_REQUEST,
    BUS_WAIT
} wishbone_state;

wishbone_state state, next_state;
assign state_out = state;
logic [ADDR_WIDTH-1:0] addr_reg;
logic [DATA_WIDTH-1:0] data_reg;
logic we_reg;
logic [SEL_WIDTH-1:0] sel_reg;
logic stb_reg;
logic cyc_reg;
logic [DATA_WIDTH-1:0] data_received_reg;
logic ack_reg;
logic [2:0] cti_reg;
logic err_reg;

```

```

logic tag_add_reg;
assign addr_o = addr_reg;
assign data_o = data_reg;
assign we_o = we_reg;
assign sel_o = sel_reg;
assign stb_o = stb_reg;
assign cyc_o = cyc_reg;
assign cti_o = cti_reg;
assign tag_add_out2s = tag_add_reg;
assign data_received = data_received_reg;
assign acknowledgement = ack_reg;
assign error = err_reg;

always_ff @(posedge clk_i or posedge rst_i) begin
    if (rst_i)
        state <= IDLE;
    else
        state <= next_state;
end

always_comb begin
    case (state)
        IDLE : begin
            if(cyc_input & strobe)
                next_state = BUS_REQUEST;
            else
                next_state = IDLE;
        end
        BUS_REQUEST : begin
            if(cyc_input & !strobe & (((cti_input[2:0] == 3'b001)) | ((cti_input[2:0] == 3'b010)))) &
(ack_reg|err_reg))
                next_state = BUS_WAIT;
            else if (!cyc_input & !strobe & ((cti_input[2:0] == 3'b000)) & (ack_reg|err_reg))
                next_state = IDLE;
            else
                next_state = BUS_REQUEST;
        end
        BUS_WAIT : begin
            if(cyc_input & strobe & (((cti_input[2:0] == 3'b001)) | ((cti_input[2:0] == 3'b010))))
                next_state = BUS_REQUEST;
            else if(!cyc_input & !strobe & (cti_input[2:0] == 3'b111))
                next_state = IDLE;
            else
                next_state = BUS_WAIT;
        end
        default : next_state = IDLE;
    endcase
end

always_ff @(posedge clk_i or posedge rst_i) begin

```

```

if (rst_i) begin
    addr_reg <= {ADDR_WIDTH{1'b0}};
    data_reg <= {DATA_WIDTH{1'b0}};
    we_reg <= 1'b0;
    sel_reg <= {SEL_WIDTH{1'b0}};
    stb_reg <= 1'b0;
    cyc_reg <= 1'b0;
    tag_add_reg <= 1'b0;
    data_received_reg <= {DATA_WIDTH{1'b0}};
    ack_reg <= 1'b0;
    cti_reg <= 3'b000;
    err_reg <= 1'b0;
end else begin
    case (next_state)
        IDLE: begin
            addr_reg <= {ADDR_WIDTH{1'b0}};
            data_reg <= {DATA_WIDTH{1'b0}};
            we_reg <= 1'b0;
            sel_reg <= {SEL_WIDTH{1'b0}};
            stb_reg <= 1'b0;
            cyc_reg <= 1'b0;
            ack_reg <= 1'b0;
            data_received_reg <= {DATA_WIDTH{1'b0}};
            tag_add_reg <= 1'b0;
            cti_reg <= 3'b000;
            err_reg <= 1'b0;
        end
        BUS_REQUEST: begin
            addr_reg <= addr_input;
            sel_reg <= sel_input;
            we_reg <= write_enable;
            stb_reg <= strobe;
            cyc_reg <= cyc_input;
            ack_reg <= ack_i;
            err_reg <= err_i;
            cti_reg <= cti_input;
            tag_add_reg <= tag_add_i;
            if (write_enable)
                data_reg <= data_input;
            else if (!write_enable)
                data_received_reg <= data_i;
        end
        BUS_WAIT: begin
            addr_reg <= {ADDR_WIDTH{1'b0}};
            data_reg <= {DATA_WIDTH{1'b0}};
            sel_reg <= {SEL_WIDTH{1'b0}};
            data_received_reg <= {DATA_WIDTH{1'b0}};
            stb_reg <= 1'b0;
            cyc_reg <= cyc_input;
            we_reg <= we_reg;

```

```

        tag_add_reg <= 1'b0;
        ack_reg <= 1'b0;
        cti_reg <= 3'b000;
        err_reg <= 1'b0;
    end
endcase
end
end
endmodule

module wishbone_slave #(
parameter int ADDR_WIDTH = 5,
parameter int DATA_WIDTH = 32,
parameter int SEL_WIDTH = DATA_WIDTH/8
)(
input logic rst_i,
input logic clk_i,
input logic [ADDR_WIDTH-1:0] addr_i,
input logic [DATA_WIDTH-1:0] data_i,
input logic we_i,
input logic [SEL_WIDTH-1:0] sel_i,
input logic stb_i,
input logic cyc_i,
input logic [2:0] cti_i,
input logic tag_add_i,
output logic ack_o,
output logic err_o,
output logic [DATA_WIDTH-1:0] data_o,
output logic [ADDR_WIDTH-1:0] counter_out);
logic [DATA_WIDTH-1:0] mem [0:(2**ADDR_WIDTH)-1];
logic [ADDR_WIDTH-1:0] counter;
assign counter_out = counter;

always_ff @(posedge clk_i or posedge rst_i) begin
    if (rst_i) begin
        ack_o <= 1'b0;
        err_o <= 1'b0;
        data_o <= {DATA_WIDTH{1'b0}};
        counter <= {ADDR_WIDTH{1'b0}};
    end
    else begin
        if (!cyc_i && !stb_i) begin
            counter <= {ADDR_WIDTH{1'b0}};
        end
        else if (stb_i & cyc_i & we_i) begin
            if (cti_i == 3'b001 | cti_i == 3'b000) begin
                if ((addr_i + counter) > 15) begin
                    err_o <= 1'b1;
                    ack_o <= 1'b0;
                end
            end
        end
    end
end

```

```

else begin
    for (int i = 0; i < SEL_WIDTH; i++) begin
        if (sel_i[i] == 1'b1) begin
            mem[addr_i][(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)] <=
data_i[(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)];
            end
        end
        ack_o <= ( stb_i & cyc_i );
        err_o <= 1'b0;
        data_o <= {DATA_WIDTH{1'b0}};
        end
    end
else if (cti_i == 3'b010) begin
    if ((addr_i + counter) > 15) begin
        err_o <= 1'b1;
        ack_o <= 1'b0;
        counter <= counter + 1;
        end
    else begin
        counter <= counter + 1;
        for (int i = 0; i < SEL_WIDTH; i++) begin
            if (sel_i[i] == 1'b1) begin
                mem[addr_i + counter][(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)]
<= data_i[(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)];
                end
            end
            ack_o <= ( stb_i & cyc_i );
            err_o <= 1'b0;
            data_o <= {DATA_WIDTH{1'b0}};
            end
        end
    end
else if (stb_i & cyc_i & !we_i) begin
    if(tag_add_i) begin
        ack_o <= ( stb_i & cyc_i );
        err_o <= 1'b0;
        data_o <= mem[0] + mem[1];
        end
    else if (cti_i == 3'b001 | cti_i == 3'b000) begin
        if ((addr_i + counter) > 15) begin
            err_o <= 1'b1;
            ack_o <= 1'b0;
            end
        else begin
            data_o <= {DATA_WIDTH{1'b0}};
            for (int i = 0; i < SEL_WIDTH; i++) begin
                if (sel_i[i] == 1'b1) begin
                    data_o[(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)] <=
mem[addr_i][(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)];
                    end
                end
            end
        end
    end
end

```

```

        end
        ack_o <= ( stb_i & cyc_i );
        err_o <= 1'b0;
    end
end
else if (cti_i == 3'b010) begin
    if ((addr_i + counter) > 15) begin
        err_o <= 1'b1;
        ack_o <= 1'b0;
        data_o <= {DATA_WIDTH{1'b0}};
        counter <= counter + 1;
    end
    else begin
        counter <= counter + 1;
        for (int i = 0; i < SEL_WIDTH; i++) begin
            if (sel_i[i]) begin
                data_o[(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)] <=
mem[addr_i + counter][(i+1)*(DATA_WIDTH/SEL_WIDTH)-1 : (DATA_WIDTH/SEL_WIDTH)];
            end
        end
        ack_o <= ( stb_i & cyc_i );
        err_o <= 1'b0;
    end
end
end
else begin
    ack_o <= 1'b0;
    err_o <= 1'b0;
    data_o <= {DATA_WIDTH{1'b0}};
end
end
end
endmodule

```

## 5.2 Testbench Code:

```

//=====
// Interface
//=====
interface wishbone_if #(parameter ADDR_WIDTH = 5, DATA_WIDTH = 32, SEL_WIDTH =
DATA_WIDTH/8)(input logic clk);
    logic rst;
    logic [ADDR_WIDTH-1:0] addr_o;
    logic [DATA_WIDTH-1:0] data_o;
    logic we_o;
    logic stb_o;
    logic cyc_o;
    logic [SEL_WIDTH-1:0] sel_o;
    logic [2:0] cti_input;

```

```

logic tag_add;

logic ack_i;
logic err_i;
logic [DATA_WIDTH-1:0] data_i;
logic [1:0] state_out;
logic [ADDR_WIDTH-1:0] counter_dbg;
endinterface

//=====================================================
// Transaction Class
//=====================================================
class transaction;
    rand bit [4:0] addr_o;
    rand bit [31:0] data_o;
    rand bit [3:0] sel_o;
    rand bit we_o;
    bit [31:0] data_i;
    bit ack_i;
    bit err_i;
    rand bit tag_add;
    rand bit [2:0] cti_input;
    bit stb_o;
    bit cyc_o;
    function void formatted_display(string tag);
        $display("# -----");
        $display("# - [ %s ] ", tag);
        $display("# -----");
        $display("# - addr = %oh, data_out = %oh, we = %ob, sel = %oh, tag_add = %ob, cti = %oh",
            addr_o, data_o, we_o, sel_o, tag_add, cti_input);
        $display("# - data_in = %oh, ack = %ob, err = %ob", data_i, ack_i, err_i);
        $display("# -----\\n");
    endfunction
endclass

//=====================================================
// Generator
//=====================================================
class generator;
    mailbox #(transaction) gen2drv;
    int repeat_count;
    event ended;
    function new(mailbox #(transaction) gen2drv);
        this.gen2drv = gen2drv;
    endfunction
    task run();
        transaction tr;

        // -----
        // PHASE 1: WRITE transactions

```

```

// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 1: WRITE Transactions (we=1)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == i;
        cti_input == 3'b000;
        sel_o == 4'b1111;
        we_o == 1'b1;
        tag_add == 1'b0;
    }) else $error("Randomization failed for CLASSIC WRITE transaction");
    tr.formatted_display($sformatf("Generator - CLASSIC WRITE (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----
// PHASE 2: READ transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 2: READ Transactions (we=0)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == i;
        data_o == 32'b0;
        cti_input == 3'b000;
        sel_o == 4'b1111;
        we_o == 1'b0;
        tag_add == 1'b0;
    }) else $error("Randomization failed for CLASSIC READ transaction");
    tr.formatted_display($sformatf("Generator - CLASSIC READ (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----
// PHASE 3: TAG READ transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 3: TAG ADD Transactions (we=0 & tag_add=1)");
$display("# ----- \n");
tr = new();
assert (tr.randomize() with {
    addr_o == 32'b0;
    data_o == 32'b0;
    cti_input == 3'b000;

```

```

sel_o == 4'b1111;
we_o == 1'b0;
tag_add == 1'b1;
}) else $error("Randomization failed for TAG READ transaction");
tr.formatted_display("Generator - TAG ADD READ");
gen2drv.put(tr);
repeat_count++;

// -----
// PHASE 4: BLOCK WRITE transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 4: BLOCK WRITE (we=1 & cti = 001)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == i;
        cti_input == 3'b001;
        sel_o == 4'b1111;
        we_o == 1'b1;
        tag_add == 1'b0;
    }) else $error("Randomization failed for BLOCK WRITE transaction");
    tr.formatted_display($sformatf("Generator - BLOCK WRITE (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----
// PHASE 5: BLOCK READ transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 4: BLOCK READ (we=0 & cti = 001)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == i;
        data_o == 32'b0;
        cti_input == 3'b001;
        sel_o == 4'b1111;
        we_o == 1'b0;
        tag_add == 1'b0;
    }) else $error("Randomization failed for BLOCK READ transaction");
    tr.formatted_display($sformatf("Generator - BLOCK READ (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----

```

```

// PHASE 6: TAG READ transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 6: TAG ADD Transactions (we=0 & tag_add=1)");
$display("# ----- \n");
tr = new();
assert (tr.randomize() with {
    addr_o == 32'b0;
    data_o == 32'b0;
    cti_input == 3'b001;
    sel_o == 4'b1111;
    we_o == 1'b0;
    tag_add == 1'b1;
}) else $error("Randomization failed for TAG READ transaction");
tr.formatted_display("Generator - TAG ADD READ");
gen2drv.put(tr);
repeat_count++;

// -----
// PHASE 7: BLOCK END transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 7: BLOCK END (we=1 & cti = 111)");
$display("# ----- \n");
tr = new();
assert (tr.randomize() with {
    cti_input == 3'b111;
    sel_o == 4'b1111;
    we_o == 1'b1;
    tag_add == 1'b0;
}) else $error("Randomization failed for BLOCK WRITE END transaction");
tr.formatted_display("Generator - BLOCK WRITE END");
gen2drv.put(tr);
repeat_count++;

// -----
// PHASE 8: BLOCK WRITE transactions CTI = 010
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 7: BLOCK WRITE (we=1 & cti = 010)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == 5'b00000;
        cti_input == 3'b010;
        sel_o == 4'b1111;
        we_o == 1'b1;
        tag_add == 1'b0;
    }) else $error("Randomization failed for BLOCK WRITE CTI = 010 transaction");
end

```

```

    tr.formatted_display($sformatf("Generator - BLOCK WRITE FOR THE CTI = 010 (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----
// PHASE 9: BLOCK END transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 7: BLOCK END (we=1 & cti = 111)");
$display("# ----- \n");
tr = new();
assert (tr.randomize() with {
    cti_input == 3'b111;
    sel_o == 4'b1111;
    we_o == 1'b1;
    tag_add == 1'b0;
}) else $error("Randomization failed for BLOCK WRITE END transaction");
tr.formatted_display("Generator - BLOCK WRITE END");
gen2drv.put(tr);
repeat_count++;

// -----
// PHASE 10: BLOCK READ transactions CTI = 010
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 10: BLOCK WRITE (we=1 & cti = 010)");
$display("# ----- \n");
for (int i = 0; i < 2; i++) begin
    tr = new();
    assert (tr.randomize() with {
        addr_o == 5'b00000;
        cti_input == 3'b010;
        sel_o == 4'b1111;
        we_o == 1'b0;
        tag_add == 1'b0;
    }) else $error("Randomization failed for BLOCK WRITE CTI = 010 transaction");
    tr.formatted_display($sformatf("Generator - BLOCK WRITE FOR THE CTI = 010 (addr=%0d)", i));
    gen2drv.put(tr);
    repeat_count++;
end

// -----
// PHASE 12: TAG READ transactions
// -----
$display("\n# -----");
$display("# [GENERATOR] PHASE 11: TAG ADD Transactions (we=0 & tag_add=1)");
$display("# ----- \n");
tr = new();
assert (tr.randomize() with {

```

```

    addr_o == 32'b0;
    data_o == 32'b0;
    cti_input == 3'b010;
    sel_o == 4'b1111;
    we_o == 1'b0;
    tag_add == 1'b1;
  }) else $error("Randomization failed for TAG READ transaction");
  tr.formatted_display("Generator - TAG ADD READ");
  gen2drv.put(tr);
  repeat_count++;

  // -----
  // PHASE 13: BLOCK END transactions
  // -----
  $display("\n# -----");
  $display("# [GENERATOR] PHASE 12: BLOCK END (we=1 & cti = 111)");
  $display("# ----- \n");
  tr = new();
  assert (tr.randomize() with {
    cti_input == 3'b111;
    sel_o == 4'b1111;
    we_o == 1'b0;
    tag_add == 1'b0;
  }) else $error("Randomization failed for BLOCK WRITE END transaction");
  tr.formatted_display("Generator - BLOCK WRITE END");
  gen2drv.put(tr);
  repeat_count++;
  ->ended;
  $display("# [GENERATOR] Finished sending %0d transactions", repeat_count);
endtask
endclass

//=====
// Driver
//=====
class driver;
  // Virtual interface
  virtual wishbone_if vif;
  // Mailbox to get transactions from generator
  mailbox #(transaction) gen2drv;
  int no_transactions;
  // Constructor
  function new(virtual wishbone_if vif, mailbox #(transaction) gen2drv);
    this.vif = vif;
    this.gen2drv = gen2drv;
  endfunction
  //=====
  // Reset task: set all signals to 0
  //=====
  task reset();

```

```

vif.addr_o  <= 0;
vif.data_o  <= 0;
vif.sel_o   <= 0;
vif.we_o    <= 0;
vif.stb_o   <= 0;
vif.cyc_o   <= 0;
vif.tag_add <= 0;
vif.cti_input <= 0;
wait(!vif.rst);
$display("# Driver: All DUT signals reset to 0\n");
endtask
//=====
// Main driver task
//=====
task run();
transaction tr;
forever begin
    // Wait for a new transaction from the generator
    gen2drv.get(tr);
    vif.addr_o  <= tr.addr_o;
    vif.data_o  <= tr.data_o;
    vif.sel_o   <= tr.sel_o;
    vif.we_o    <= tr.we_o;
    vif.stb_o   <= 1'b1;
    vif.cyc_o   <= 1'b1;
    vif.tag_add <= tr.tag_add;
    vif.cti_input <= tr.cti_input;
    if (tr.cti_input == 3'b000) begin
        @(posedge vif.clk);
        vif.stb_o <= 0;
        do @(posedge vif.clk); while (vif.ack_i !== 1'b1 && vif.err_i !== 1'b1);
        vif.cyc_o <= 0;
        tr.formatted_display("Driver");
        no_transactions++;
    end
    else if (tr.cti_input inside {3'b001, 3'b010}) begin
        @(posedge vif.clk);
        vif.stb_o <= 0;
        do @(posedge vif.clk); while (vif.ack_i !== 1'b1 && vif.err_i !== 1'b1);
        tr.formatted_display("Driver");
        no_transactions++;
    end
    else if (tr.cti_input == 3'b111) begin
        @(posedge vif.clk);
        @(posedge vif.clk);
        @(posedge vif.clk);
        vif.stb_o <= 0;
        vif.cyc_o <= 0;
        @(posedge vif.clk);
        @(posedge vif.clk);
    end
end

```

```

        tr.formatted_display("Driver");
        no_transactions++;
    end
end
endtask
endclass

//=====
// Monitor
//=====
class monitor;
    virtual wishbone_if vif;
    mailbox #(transaction) mon2sb;
    function new(virtual wishbone_if vif, mailbox #(transaction) mon2sb);
        this.vif = vif;
        this.mon2sb = mon2sb;
    endfunction
//=====
// Main monitor task
//=====
    task run();
        transaction tr;
        bit exit_logged = 0; // <== new flag
        forever begin
            @(posedge vif.clk);
            // -----
            // CASE 1: CTI = 000, 001, 010
            // -----
            if (vif.cti_input inside {3'b000, 3'b001, 3'b010}) begin
                exit_logged = 0;
                if (vif.ack_i || vif.err_i) begin
                    tr = new();
                    // If CTI = 010 → use incremented address (addr + counter_dbg)
                    if (vif.cti_input == 3'b010)
                        tr.addr_o = vif.addr_o + vif.counter_dbg - 1;
                    else
                        tr.addr_o = vif.addr_o;
                    tr.data_o = vif.data_o;
                    tr.sel_o = vif.sel_o;
                    tr.we_o = vif.we_o;
                    tr.data_i = vif.data_i;
                    tr.ack_i = vif.ack_i;
                    tr.err_i = vif.err_i;
                    tr.tag_add = vif.tag_add;
                    tr.cti_input = vif.cti_input;

                    tr.formatted_display("Monitor - CTI=000/001/010");
                    mon2sb.put(tr);
                end
            end
        end
    end
end

```

```

// -----
// CASE 2: CTI = 111 (End of burst)
// -----
else if (vif.cti_input == 3'b111 & !exit_logged) begin
  if (vif.stb_o == 0 && vif.cyc_o == 0) begin
    tr = new();
    tr.addr_o = vif.addr_o;
    tr.data_o = vif.data_o;
    tr.sel_o = vif.sel_o;
    tr.we_o = vif.we_o;
    tr.data_i = vif.data_i;
    tr.ack_i = vif.ack_i;
    tr.err_i = vif.err_i;
    tr.tag_add = vif.tag_add;
    tr.cti_input = vif.cti_input;
    tr.cyc_o = vif.cyc_o;
    tr.stb_o = vif.stb_o;

    tr.formatted_display("Monitor - CTI - 111");
    mon2sb.put(tr);
    exit_logged = 1;
  end
end
endtask
endclass

//=====
// Scoreboard
//=====
class scoreboard;
  mailbox #(transaction) mon2sb;
  int no_transactions;
  bit [31:0] exp_mem [0:15];
  function new(mailbox #(transaction) mon2sb);
    this.mon2sb = mon2sb;
    // Initialize memory contents
    foreach (exp_mem[i])
      exp_mem[i] = '0;
  endfunction
  task run();
    transaction tr;
    forever begin
      mon2sb.get(tr);
      // -----
      // BLOCK WRITE OR READ EXIT MODE VERIFY
      // -----
      if ( tr.cti_input == 3'b111 ) begin
        if ( tr.cyc_o == 1'bo && tr.stb_o == 1'bo )
          $display("# BLOCK EXIT DONE ");
      end
    end
  endtask
endclass

```

```

else
    $display("# BLOCK EXIT NOT PERFORMED DONE ");
end
// -----
// WRITE operation
// -----
else if (tr.we_o == 1'b1 && tr.tag_add == 1'bo) begin
    if (tr.addr_o < 5'd16) begin
        if (tr.ack_i && !tr.err_i) begin
            exp_mem[tr.addr_o] = tr.data_o;
            $display("# [WRITE][PASS] Addr=%0d Data=0x%0h stored ", tr.addr_o, tr.data_o);
        end else begin
            $display("# [WRITE][FAIL] Addr=%0d Expected ACK=1 ERR=0 but got ACK=%0b ERR=%0b ",
                tr.addr_o, tr.ack_i, tr.err_i);
        end
    end else begin
        if (tr.err_i && !tr.ack_i)
            $display("# [WRITE][PASS] Addr=%0d Invalid access handled (ERR=1,ACK=0) ", tr.addr_o);
        else
            $display("# [WRITE][FAIL] Addr=%0d Invalid access not handled correctly ", tr.addr_o);
        end
    end
end
// -----
// TAG ADD READ
// -----
else if (tr.we_o == 1'bo && tr.tag_add == 1'b1) begin
    bit [31:0] exp_tag_sum = exp_mem[0] + exp_mem[1];
    if (tr.data_i !== exp_tag_sum)
        $error(" TAG ADD MISMATCH -> exp=%h got=%h",
            exp_tag_sum, tr.data_i);
    else
        $display(" TAG ADD OK -> SUM(%h + %h) = %h",
            exp_mem[0], exp_mem[1], tr.data_i);
    end
// -----
// READ operation (compare data)
// -----
else begin
    if (tr.addr_o < 5'd16) begin
        if (tr.ack_i && !tr.err_i) begin
            if (tr.data_i === exp_mem[tr.addr_o])
                $display("# [READ][PASS] Addr=%0d Data match (Expected=0x%0h, Got=0x%0h)",
                    tr.addr_o, exp_mem[tr.addr_o], tr.data_i);
            else
                $display("# [READ][FAIL] Addr=%0d Data mismatch (Expected=0x%0h, Got=0x%0h)",
                    tr.addr_o, exp_mem[tr.addr_o], tr.data_i);
        end else begin
            $display("# [READ][FAIL] Addr=%0d Expected ACK=1 ERR=0 but got ACK=%0b ERR=%0b ",
                tr.addr_o, tr.ack_i, tr.err_i);
        end
    end
end

```

```

end else begin
  if (tr.err_i && !tr.ack_i)
    $display("# [READ][PASS] Addr=%0d Invalid access handled (ERR=1,ACK=0) ", tr.addr_o);
  else
    $display("# [READ][FAIL] Addr=%0d Invalid access not handled correctly ", tr.addr_o);
  end
end
no_transactions++;
if (tr.cti_input != 3'b111)
  tr.formatted_display("Scoreboard");
end
endtask
endclass

```

```

//=====
// TEST CLASS (Top-level Verification Orchestrator)
//=====
class environment;
  generator gen;
  driver driv;
  monitor mon;
  scoreboard scb;
  mailbox #(transaction) gen2drv;
  mailbox #(transaction) mon2scb;
  virtual wishbone_if vif;
  function new(virtual wishbone_if vif);
    this.vif = vif;
    gen2drv = new();
    mon2scb = new();
    gen = new(gen2drv);
    driv = new(vif, gen2drv);
    mon = new(vif, mon2scb);
    scb = new(mon2scb);
  endfunction
  task pre_test();
    driv.reset();
  endtask
  task test();
    fork
      gen.run();
      driv.run();
      mon.run();
      scb.run();
    join_any
  endtask
  task post_test();
    wait(gen.ended.triggered);
    wait(gen.repeat_count == driv.no_transactions);
    wait(gen.repeat_count == scb.no_transactions);
    $display("\n# =====");
  endtask
endclass

```

```

    $display("# TEST COMPLETE : All transactions processed!");
    $display("# Generator: %0d | Driver: %0d | Scoreboard: %0d",
        gen.repeat_count, driv.no_transactions, scb.no_transactions);
    $display("# =====\n");
endtask
task run();
    pre_test();
    test();
    post_test();
    $finish;
endtask
endclass

```

```

program test(wishbone_if i_intf);
    environment env;
    initial begin
        env = new(i_intf);
        env.run();
    end
endprogram

```

```

//=====
// Top Testbench Module
//=====
module testbench;
    parameter ADDR_WIDTH = 5;
    parameter DATA_WIDTH = 32;
    parameter SEL_WIDTH = DATA_WIDTH/8;
    logic clk = 0;
    logic rst = 0;
    always #2 clk = ~clk;
    wishbone_if #(ADDR_WIDTH, DATA_WIDTH, SEL_WIDTH) i_intf(clk);
    test tb_inst(.i_intf(i_intf));
    wishbone_top #(
        .ADDR_WIDTH(ADDR_WIDTH),
        .DATA_WIDTH(DATA_WIDTH),
        .SEL_WIDTH(SEL_WIDTH)
    ) dut (
        .clk_i(clk),
        .rst_i(i_intf.rst),
        .addr_o(i_intf.addr_o),
        .data_o(i_intf.data_o),
        .we_o(i_intf.we_o),
        .stb_o(i_intf.stb_o),
        .cyc_o(i_intf.cyc_o),
        .sel_o(i_intf.sel_o),
        .cti_input(i_intf.cti_input),
        .tag_add(i_intf.tag_add),
        .ack_i(i_intf.ack_i),
        .err_i(i_intf.err_i),

```

```

.data_i(i_intf.data_i),
.state_out(i_intf.state_out),
.dbg_w_addr(),
.dbg_w_data_m2s(),
.counter(i_intf.counter_dbg),
.dbg_w_we(),
.dbg_tag_add(),
.dbg_w_sel(),
.dbg_w_stb(),
.dbg_w_cyc(),
.dbg_w_ack(),
.dbg_w_err(),
.dbg_cti() );
initial begin
    $dumpfile("wishbone_wave.vcd");
    $dumpvars(0, testbench);
    i_intf.rst = 1;
    repeat(5) @(posedge clk);
    i_intf.rst = 0;
end
endmodule

```

## CHAPTER 6

### OUTPUT

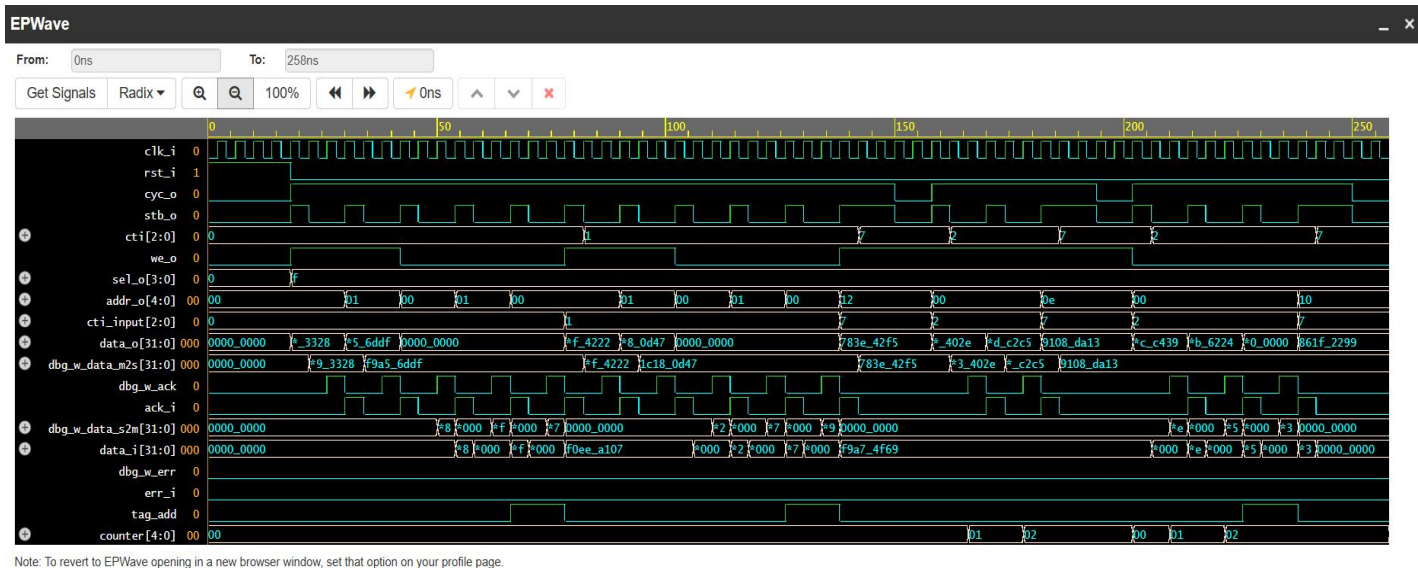


Fig 3: Waveform

```

# Driver: All DUT signals reset to 0

# -----
# [GENERATOR] PHASE 1: WRITE Transactions (we=1)
# -----
# -----
# - [ Generator - CLASSIC WRITE (addr=0) ]
# -----
# - addr = 0, data_out = f7493328, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - CLASSIC WRITE (addr=1) ]
# -----
# - addr = 1, data_out = f9a56ddf, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 2: READ Transactions (we=0)
# -----
# -----
# - [ Generator - CLASSIC READ (addr=0) ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - CLASSIC READ (addr=1) ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 3: TAG ADD Transactions (we=0 & tag_add=1)
# -----
# -----
# - [ Generator - TAG ADD READ ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 4: BLOCK WRITE (we=1 & cti = 001)
# -----
# -----

```

```

# - [ Generator - BLOCK WRITE (addr=0) ]
# -----
# - addr = 0, data_out = dd8f4222, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - BLOCK WRITE (addr=1) ]
# -----
# - addr = 1, data_out = 1c180d47, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 4: BLOCK READ (we=0 & cti = 001)
# -----
# -----
# - [ Generator - BLOCK READ (addr=0) ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - BLOCK READ (addr=1) ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 6: TAG ADD Transactions (we=0 & tag_add=1)
# -----
# -----
# - [ Generator - TAG ADD READ ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 7: BLOCK END (we=1 & cti = 111)
# -----
# -----
# - [ Generator - BLOCK WRITE END ]
# -----
# - addr = 12, data_out = 783e42f5, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# -----

```

```

# [GENERATOR] PHASE 7: BLOCK WRITE (we=1 & cti = 010)
# -----
# -----
# - [ Generator - BLOCK WRITE FOR THE CTI = 010 (addr=0) ]
# -----
# - addr = 0, data_out = 4c13402e, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - BLOCK WRITE FOR THE CTI = 010 (addr=1) ]
# -----
# - addr = 0, data_out = a8ddc2c5, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 7: BLOCK END (we=1 & cti = 111)
# -----
# -----
# - [ Generator - BLOCK WRITE END ]
# -----
# - addr = e, data_out = 9108da13, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 10: BLOCK WRITE (we=1 & cti = 010)
# -----
# -----
# - [ Generator - BLOCK WRITE FOR THE CTI = 010 (addr=0) ]
# -----
# - addr = 0, data_out = 6e4cc439, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Generator - BLOCK WRITE FOR THE CTI = 010 (addr=1) ]
# -----
# - addr = 0, data_out = fa6b6224, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 11: TAG ADD Transactions (we=0 & tag_add=1)
# -----
# -----
# - [ Generator - TAG ADD READ ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 2

```

```

# - data_in = 0, ack = 0, err = 0
# -----
# -----
# [GENERATOR] PHASE 12: BLOCK END (we=1 & cti = 111)
# -----
# -----
# - [ Generator - BLOCK WRITE END ]
# -----
# - addr = 10, data_out = 861f2299, we = 0, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----

# [GENERATOR] Finished sending 18 transactions
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = f7493328, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = f7493328, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=0 Data=0xf7493328 stored
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = f7493328, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 1, data_out = f9a56ddf, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = f9a56ddf, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=1 Data=0xf9a56ddf stored

```

```

# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = f9a56ddf, we = 1, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = f7493328, ack = 1, err = 0
# -----
# [READ][PASS] Addr=0 Data match (Expected=0xf7493328, Got=0xf7493328)
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = f7493328, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = f9a56ddf, ack = 1, err = 0
# -----
# [READ][PASS] Addr=1 Data match (Expected=0xf9a56ddf, Got=0xf9a56ddf)
# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 0
# - data_in = f9a56ddf, ack = 1, err = 0
# -----
# -----

```

```

# - [ Driver ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 0
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 0
# - data_in = f0eea107, ack = 1, err = 0
# -----
TAG ADD OK -> SUM(f7493328 + f9a56ddf) = f0eea107
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 0
# - data_in = f0eea107, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = dd8f4222, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = dd8f4222, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = f0eea107, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=0 Data=0xdd8f4222 stored
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = dd8f4222, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = f0eea107, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 1, data_out = 1c180d47, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]

```

```

# -----
# - addr = 1, data_out = 1c180d47, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = f0eea107, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=1 Data=0x1c180d47 stored
# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = 1c180d47, we = 1, sel = f, tag_add = 0, cti = 1
# - data_in = f0eea107, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = dd8f4222, ack = 1, err = 0
# -----
# [READ][PASS] Addr=0 Data match (Expected=0xdd8f4222, Got=0xdd8f4222)
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = dd8f4222, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 1c180d47, ack = 1, err = 0
# -----
# [READ][PASS] Addr=1 Data match (Expected=0x1c180d47, Got=0x1c180d47)
# -----
# - [ Scoreboard ]

```

```

# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 0, cti = 1
# - data_in = 1c180d47, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 1
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 1
# - data_in = f9a74f69, ack = 1, err = 0
# -----
TAG ADD OK -> SUM(dd8f4222 + 1c180d47) = f9a74f69
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 1
# - data_in = f9a74f69, ack = 1, err = 0
# -----
# -----
# - [ Monitor - CTI - 111 ]
# -----
# - addr = 12, data_out = 783e42f5, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = f9a74f69, ack = 0, err = 0
# -----
# BLOCK EXIT DONE
# -----
# - [ Driver ]
# -----
# - addr = 12, data_out = 783e42f5, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = 4c13402e, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----

```

```

# - addr = 0, data_out = 4c13402e, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = f9a74f69, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=0 Data=0x4c13402e stored
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 4c13402e, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = f9a74f69, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = a8ddc2c5, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = a8ddc2c5, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = f9a74f69, ack = 1, err = 0
# -----
# [WRITE][PASS] Addr=1 Data=0xa8ddc2c5 stored
# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = a8ddc2c5, we = 1, sel = f, tag_add = 0, cti = 2
# - data_in = f9a74f69, ack = 1, err = 0
# -----
# -----
# - [ Monitor - CTI - 111 ]
# -----
# - addr = e, data_out = 9108da13, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = f9a74f69, ack = 0, err = 0
# -----
# BLOCK EXIT DONE
# -----
# - [ Driver ]
# -----
# - addr = e, data_out = 9108da13, we = 1, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Driver ]
# -----

```

```

# - addr = 0, data_out = 6e4cc439, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 0, data_out = 6e4cc439, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 4c13402e, ack = 1, err = 0
# -----
# [READ][PASS] Addr=0 Data match (Expected=0x4c13402e, Got=0x4c13402e)
# -----
# - [ Scoreboard ]
# -----
# - addr = 0, data_out = 6e4cc439, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 4c13402e, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = fa6b6224, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = fa6b6224, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = a8ddc2c5, ack = 1, err = 0
# -----
# [READ][PASS] Addr=1 Data match (Expected=0xa8ddc2c5, Got=0xa8ddc2c5)
# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = fa6b6224, we = 0, sel = f, tag_add = 0, cti = 2
# - data_in = a8ddc2c5, ack = 1, err = 0
# -----
# -----
# - [ Driver ]
# -----
# - addr = 0, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 2
# - data_in = 0, ack = 0, err = 0
# -----
# -----
# - [ Monitor - CTI=000/001/010 ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 2

```

```

# - data_in = f4f102f3, ack = 1, err = 0
# -----
TAG ADD OK -> SUM(4c13402e + a8ddc2c5) = f4f102f3
# -----
# - [ Scoreboard ]
# -----
# - addr = 1, data_out = 0, we = 0, sel = f, tag_add = 1, cti = 2
# - data_in = f4f102f3, ack = 1, err = 0
# -----
# -----
# - [ Monitor - CTI - 111 ]
# -----
# - addr = 10, data_out = 861f2299, we = 0, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# BLOCK EXIT DONE
# -----
# - [ Driver ]
# -----
# - addr = 10, data_out = 861f2299, we = 0, sel = f, tag_add = 0, cti = 7
# - data_in = 0, ack = 0, err = 0
# -----
# =====
# TEST COMPLETE : All transactions processed!
# Generator: 18 | Driver: 18 | Scoreboard: 18
# =====

```

\$finish called from file "testbench.sv", line 575.

\$finish at simulation time 258

#### V C S S i m u l a t i o n R e p o r t

Time: 258 ns

CPU Time: 0.410 seconds; Data structure size: 0.0Mb

Mon Nov 10 01:47:16 2025

Finding VCD file...

./wishbone\_wave.vcd

[2025-11-10 06:47:16 UTC] Opening EPWave...

Done

Fig 4: Log file

## **CHAPTER 7**

### **REFERENCES**

1. Sharma, M., & Kumar, D. (2011). WISHBONE Bus Architecture – A Survey and Comparison. International Journal of VLSI Design & Communication Systems (VLSICS), Vol. 3.
2. Sharma, M., & Kumar, D. (2012). Design and Synthesis of Wishbone Bus Dataflow Interface Architecture for SoC Integration. IEEE International Conference, Centre for Development of Advanced Computing (CDAC), Mohali, India.
3. Swain, A. K., & Mahapatra, K. K. (2010). Design and Verification of WISHBONE Bus Interface for System-on-Chip Integration. IEEE India Annual Conference (INDICON), National Institute of Technology, Rourkela, India.
4. Herveille, R. (2010). WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores — Revision B4. OpenCores.