I2C Master-Slave Communication Protocol

Design Specification & Verification

February 4, 2026

Table of Contents

1. Introduction

This document provides comprehensive technical documentation for a custom I2C (Inter-Integrated Circuit) master-slave communication protocol implementation. The design features 12-bit data transfer capability, clock domain crossing synchronization, and a complete SystemVerilog verification environment.

1.2 Scope

The implementation covers:

- I2C Master module with configurable clock divider
- I2C Slave module with programmable address
- 12-bit data transfer in two 8-bit transactions
- Complete verification testbench with self-checking capabilities

2. System Architecture

2.1 Overview

The I2C communication system consists of two primary components operating in a master-slave configuration. The architecture implements a synchronous design with asynchronous signal handling through proper clock domain crossing techniques.

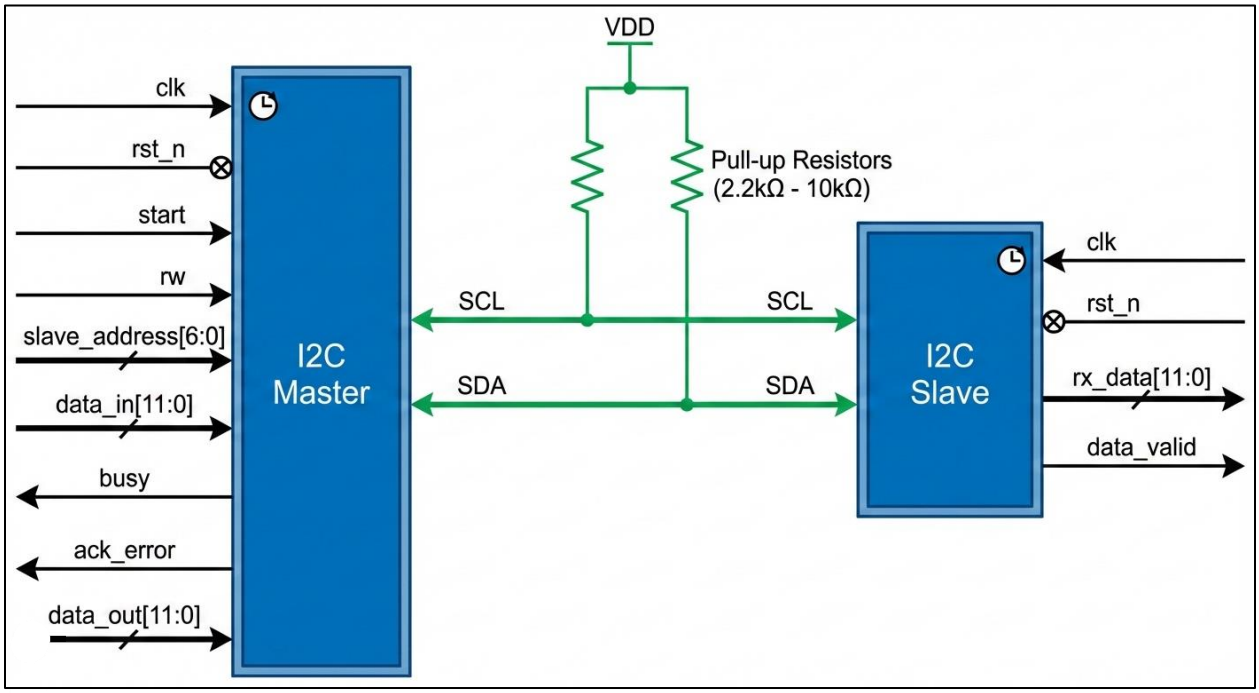

Figure 1 - System Block Diagram

2.2 Component Description

2.2.1 Master Module

The master module initiates and controls all bus transactions. Key features include:

- Configurable SCL clock generation with DIVIDER parameter
- START and STOP condition generation
- 7-bit slave addressing with R/W bit
- 12-bit data transmission in two consecutive bytes
- ACK/NACK detection and error handling
- Bidirectional data transfer support

2.2.2 Slave Module

The slave module responds to master commands and manages data transfer. Key features include:

- Programmable 7-bit slave address (SLAVE_ADDR parameter)
- START and STOP condition detection
- Address matching logic
- 12-bit internal memory for data storage
- Automatic ACK generation on address and data match
- Read and write operation support

3. Signal Description

3.1 Master Interface Signals

Signal Name	Direction	Width	Description
clk	Input	1	System clock input
rst_n	Input	1	Active-low asynchronous reset
start	Input	1	Transaction start trigger
rw	Input	1	Read (1) / Write (0) control
slave_address	Input	7	Target slave 7-bit address
data_in	Input	12	Data to write (write operations)
busy	Output	1	Transaction in progress indicator
ack_error	Output	1	ACK not received error flag
data_out	Output	12	Data read from slave (read operations)
scl	Inout	1	Serial Clock Line (open-drain)
sda	Inout	1	Serial Data Line (bidirectional)

3.2 Slave Interface Signals

Signal Name	Direction	Width	Description
clk	Input	1	System clock input
rst_n	Input	1	Active-low asynchronous reset
scl	Inout	1	Serial Clock Line
sda	Inout	1	Serial Data Line
rx_data	Output	12	Received data output
data_valid	Output	1	Data valid indicator

4. Protocol Specification

4.1 Data Format

The implementation extends standard I2C to support 12-bit data transfer. Data is transmitted in two consecutive 8-bit transactions:

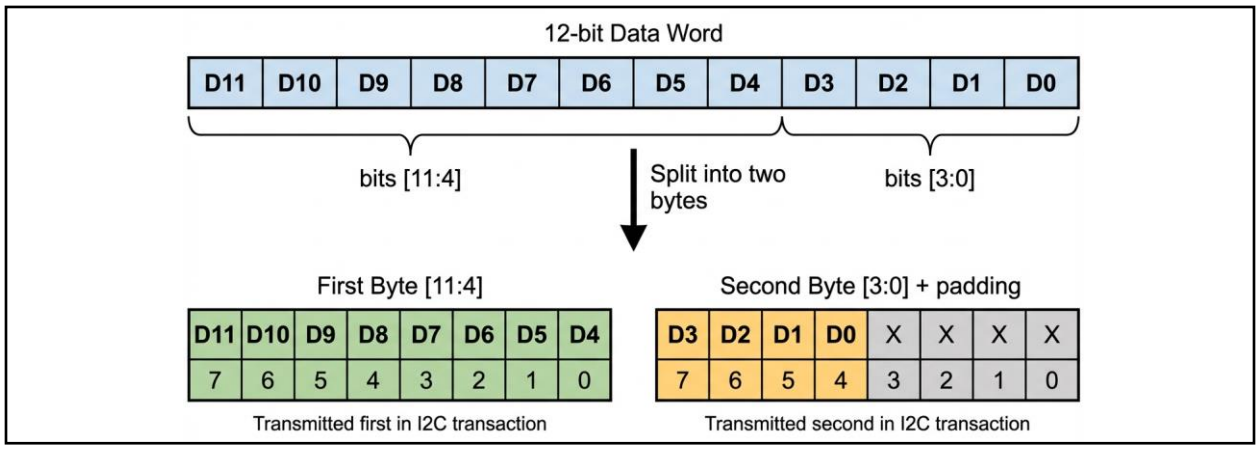

Figure 2 - 12 Bit Data Format

- **First Byte:** Contains data[11:4] (upper 8 bits)
- **Second Byte:** Contains data[3:0] in upper nibble (data[7:4]), lower nibble is don't care

4.2 Transaction Sequence

4.2.1 Write Operation

A complete write transaction consists of the following sequence:

1. **START Condition:** Master pulls SDA low while SCL is high
2. **Address Byte:** 7-bit slave address + R/W bit (0 for write)
3. **Address ACK:** Slave acknowledges address match
4. **Data Byte 1:** Upper 8 bits (data[11:4])
5. **Data ACK 1:** Slave acknowledges first byte
6. **Data Byte 2:** Lower 4 bits (data[3:0] in upper nibble)
7. **Data ACK 2:** Slave acknowledges second byte
8. **STOP Condition:** Master releases SDA to high while SCL is high

4.2.2 Read Operation

A complete read transaction consists of the following sequence:

9. **START Condition:** Master pulls SDA low while SCL is high
10. **Address Byte:** 7-bit slave address + R/W bit (1 for read)
11. **Address ACK:** Slave acknowledges address match
12. **Data Byte 1:** Slave transmits upper 8 bits (data[11:4])
13. **Master ACK:** Master acknowledges first byte (continues read)
14. **Data Byte 2:** Slave transmits lower 4 bits (data[3:0] in upper nibble)
15. **Master NACK:** Master sends NACK (terminates read)
16. **STOP Condition:** Master releases SDA to high while SCL is high.

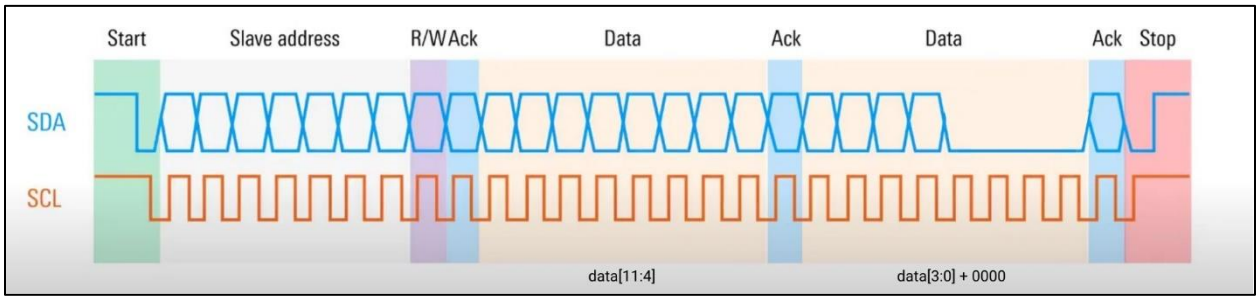

Figure 3 - Timing Diagram

5. Design Implementation

5.1 Master State Machine

The master module implements a finite state machine (FSM) to control I2C bus transactions. The state machine follows a strict separation between state register (sequential), next state logic (combinational), and output logic (sequential).

State	Description
IDLE	Waiting for transaction start
START	Generating START condition
ADDR	Transmitting address byte
ACK_ADDR	Sampling address acknowledgment
WRITE_DATA	Transmitting data byte
ACK_WRITE	Sampling write data acknowledgment
READ_DATA	Receiving data byte from slave
ACK_READ	Sending acknowledgment to slave
STOP	Generating STOP condition

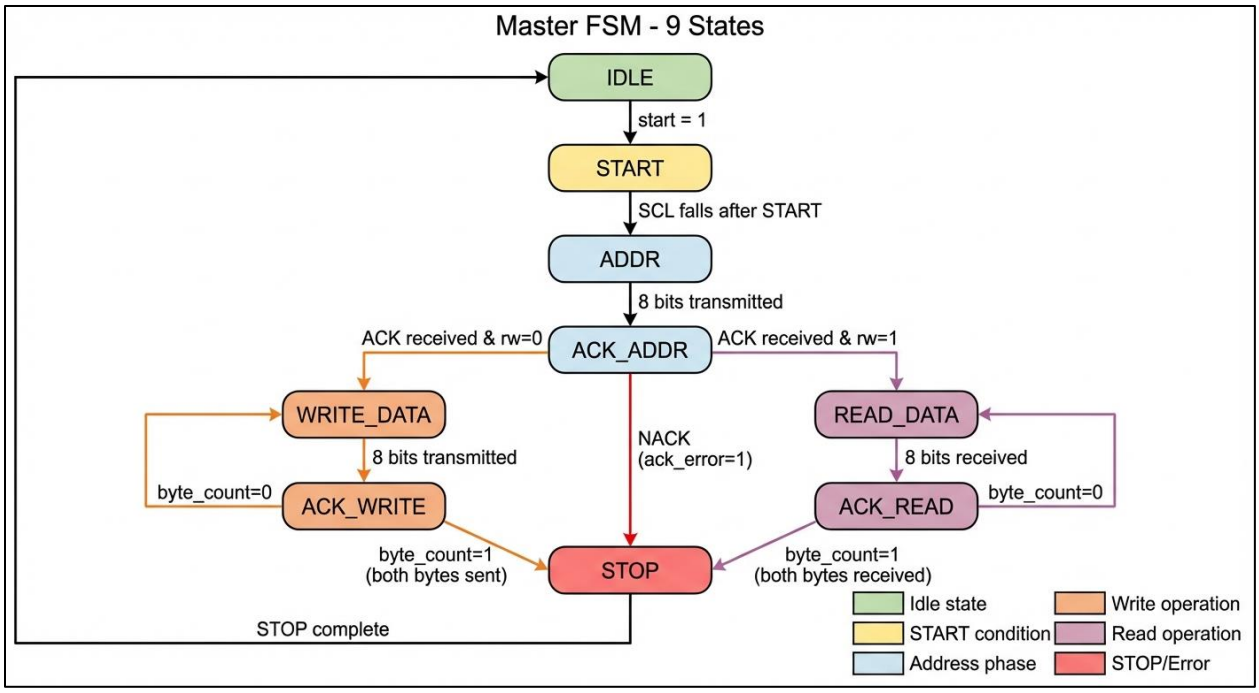

Figure 4 - Master State Machine

5.2 Slave State Machine

The slave module uses a state machine to decode I2C transactions and respond appropriately. The slave must detect START/STOP conditions and perform address matching.

State	Description
IDLE	Waiting for START condition
ADDR	Receiving and checking address byte
ACK_ADDR	Sending address acknowledgment
DATA	Transmitting or receiving data byte
ACK_DATA	Sending/receiving data acknowledgment

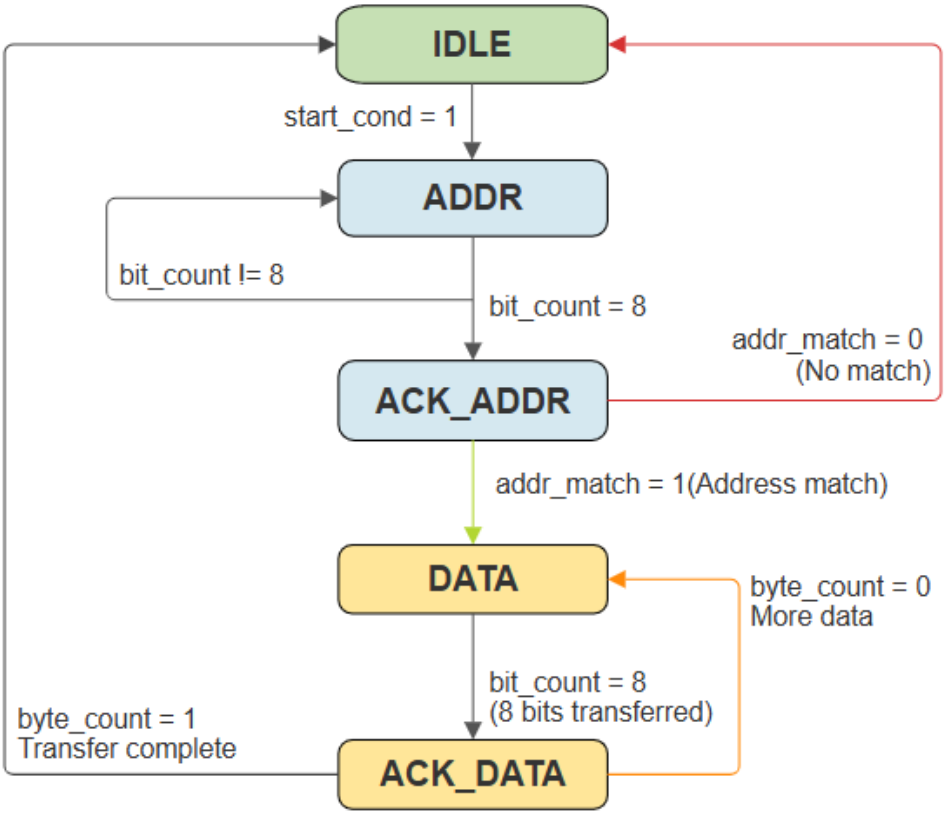

Figure 5 - Slave State Machine

5.3 Clock Domain Crossing

Both master and slave modules implement two-stage synchronizers to safely capture asynchronous I2C signals (SCL and SDA). This prevents metastability issues when crossing from the I2C clock domain to the system clock domain.

Synchronizer Implementation:

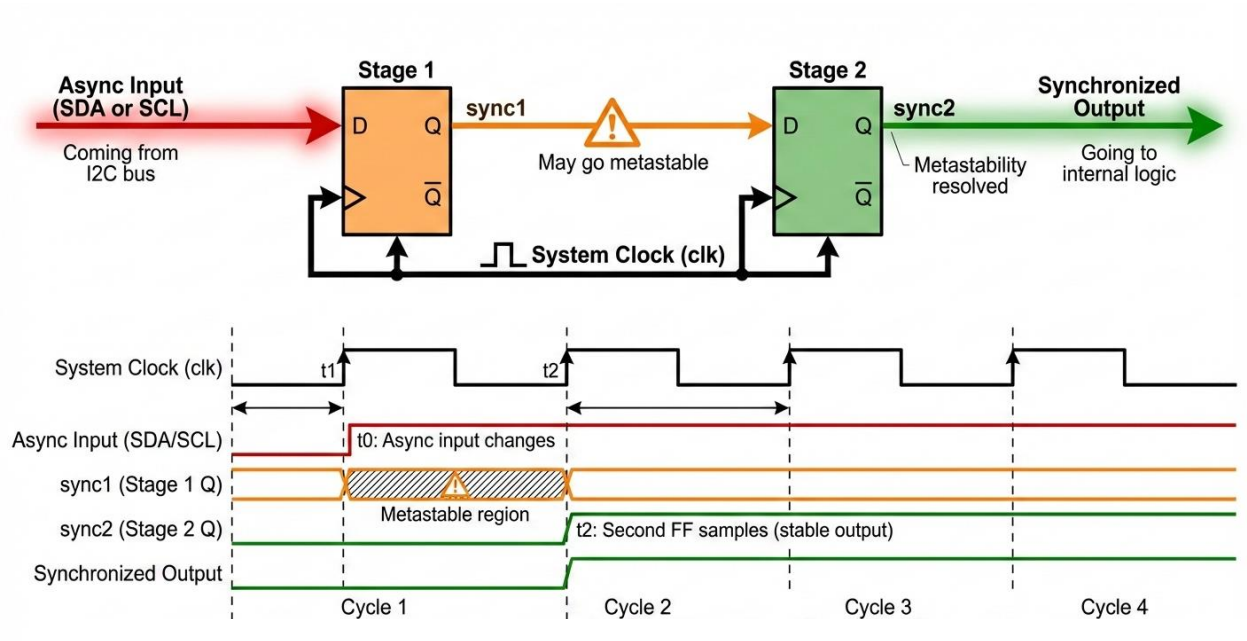

Figure 6 - Synchronizer Timing Diagram

- **First Stage:** Captures asynchronous signal; may go metastable
- **Second Stage:** Provides one clock period for metastability resolution
- Synchronized output is safe to use in logic

5.4 Tri-State Buffer Control

The SDA line requires bidirectional operation. Both master and slave use tri-state buffers controlled by output enable signals (sda_oe). The implementation follows the open-drain pattern required by I2C:

- **Drive Low:** Set sda_out=0 and sda_oe=1
- **Release (High-Z):** Set sda_oe=0, allowing external pull-up to bring line high
- Never actively drive high - use high-Z instead

5.5 SCL Clock Generation

The master generates the SCL clock by dividing the system clock. The DIVIDER parameter controls the frequency:

SCL_frequency = System_clock / (2 × DIVIDER)

For example, with a 100MHz system clock and DIVIDER=10, the SCL frequency is 5MHz. The divider uses a counter that toggles SCL when reaching DIVIDER-1.

6. Timing Specifications

6.1 Setup and Hold Times

The design ensures proper I2C timing by controlling when SDA changes relative to SCL:

Parameter	Condition	Value
SDA Setup Time	Before SCL rising edge	DIVIDER/2 clock cycles
SDA Hold Time	After SCL rising edge	DIVIDER/2 clock cycles
SCL Low Period	Half SCL period	DIVIDER clock cycles
SCL High Period	Half SCL period	DIVIDER clock cycles
START Setup Time	SDA falls while SCL high	Guaranteed by FSM
STOP Setup Time	SDA rises while SCL high	Guaranteed by FSM

6.2 Critical Timing Considerations

6.2.1 SDA Change Timing

SDA must only change when SCL is low to prevent false START/STOP conditions. The implementation ensures:

- Data bits update at scl_count=0 when scl_out is LOW
- Sampling occurs at scl_count=DIVIDER/2 when scl_out is HIGH
- Maximum setup/hold margins are maintained

6.2.2 ACK Sampling

ACK bits are sampled exactly once using a flag-based mechanism to prevent multiple sampling during the same bit period. The ack_sampled flag ensures:

- Single sample per ACK bit
- Glitch immunity
- Stable state transitions

7. Verification Environment

7.1 Testbench Architecture

The verification environment follows a layered architecture inspired by UVM (Universal Verification Methodology). Components are organized for modularity and reusability.

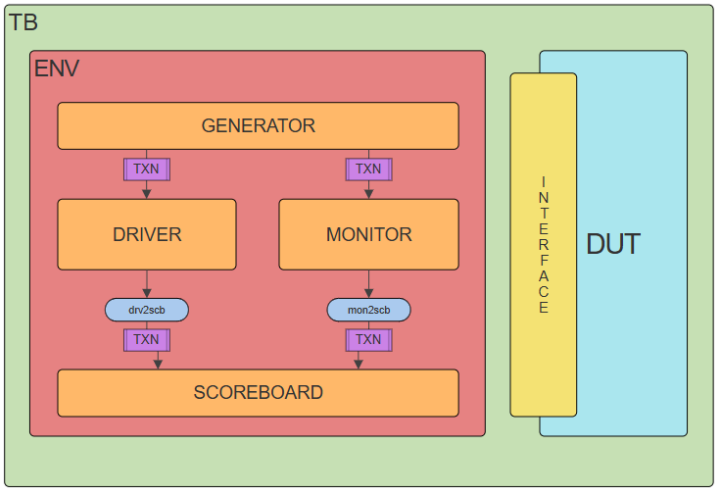

Figure 7 - Layered Testbench Architecture

7.2 Verification Components

7.2.1 Transaction Class

Defines the transaction-level model with randomizable fields and constraints:

- Read/Write control (rw)
- 7-bit slave address
- 12-bit data payload
- Error status flags

7.2.2 Generator

Creates stimulus by generating write-read transaction pairs. Each write transaction stores data in the slave, followed by a read to verify data integrity. The generator uses SystemVerilog randomization with inline constraints.

7.2.3 Driver

Converts transaction-level objects into pin-level activity. Key responsibilities:

- Wait for bus idle state
- Drive interface signals (start, rw, slave_address, data_in)
- Monitor busy flag for transaction completion
- Capture response data and error flags

7.2.4 Monitor

Observes pin-level activity and reconstructs transaction-level objects. Operates independently of the driver to provide reference checking capability.

7.2.5 Scoreboard

Implements self-checking mechanism by comparing write and read data:

- Stores written data values
- Compares read data against stored values
- Tracks pass/fail statistics
- Generates final test report

8. Parameters and Configuration

8.1 Master Module Parameters

Parameter	Type	Default	Range	Description
DIVIDER	Integer	300	2 - 65535	SCL clock divider. Controls I2C bus speed. SCL frequency = System clock / (2 × DIVIDER)

8.2 Slave Module Parameters

Parameter	Type	Default	Range	Description
SLAVE_ADDR	7-bit	7'h50	7'h00 - 7'h7F	I2C slave address. Must be unique on the bus. Reserved addresses (7'h00-7'h07, 7'h78-7'h7F) should be avoided

8.3 Configuration Examples

8.3.1 Standard Mode (100 kHz)

For a 100 MHz system clock:

```
DIVIDER = 500  // 100 MHz / (2 × 500) = 100 kHz
```

8.3.2 Fast Mode (400 kHz)

For a 100 MHz system clock:

```
DIVIDER = 125  // 100 MHz / (2 × 125) = 400 kHz
```

8.3.3 Simulation Speed

For faster simulation (used in testbench):

```
DIVIDER = 10  // 100 MHz / (2 × 10) = 5 MHz
```

9. Simulation Results

9.1 Test Configuration

The verification testbench was executed with the following configuration:

- **Total Transactions:** 20 (10 write-read pairs)
- **System Clock:** 100 MHz (10 ns period)
- **Clock Divider:** 10 (5 MHz I2C clock)
- **Slave Address:** 7'h50

9.2 Functional Verification Results

All test cases executed successfully with the following summary:

Metric	Value	Status
Total Transactions	20	✓
Write Operations	10	✓
Read Operations	10	✓
Data Integrity Checks	10	✓
ACK Errors	0	✓
Passed Tests	20	✓
Failed Tests	0	✓

```
============================================================
                TEST SUMMARY REPORT
============================================================
  Total Tests Run:    20
  Tests Passed:       20
  Tests Failed:       0
  Pass Rate:          100.0%

  ALL TESTS PASSED!

  I2C Master-Slave controller is working correctly!
============================================================
```

Status: ALL TESTS PASSED

9.3 Waveforms

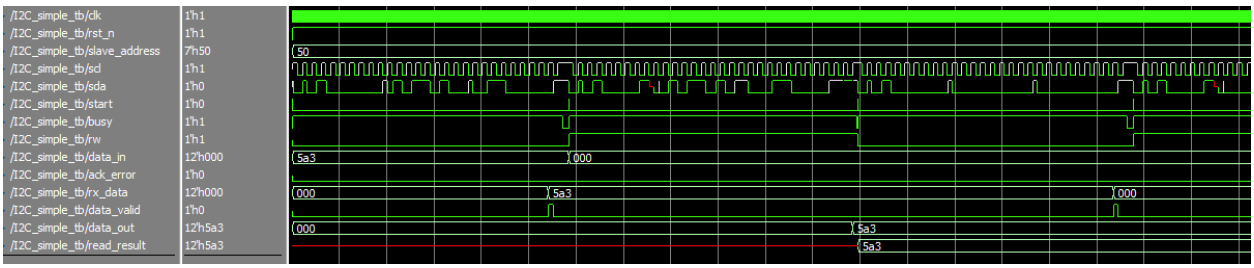

Figure 8 - 0x5A3 Write and Read Waveform

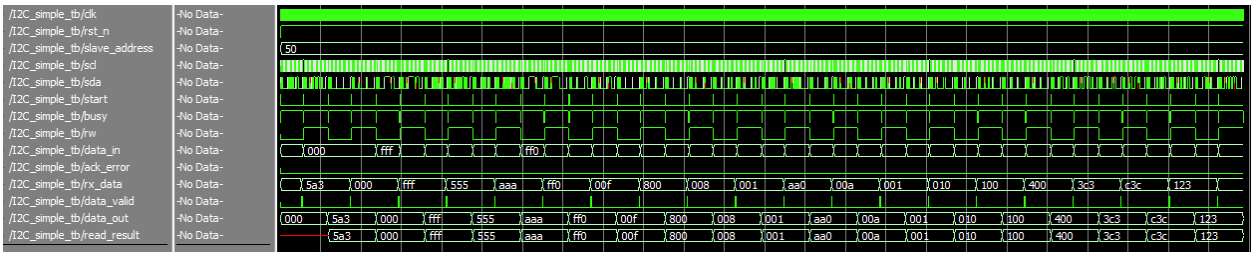

Figure 9 - All Test Waveform

9.4 Timing Analysis

Transaction timing measurements from simulation:

Operation	Duration	Clock Cycles
Single Write Transaction	5.70 µs	570 cycles
Single Read Transaction	6.10 µs	610 cycles
Write-Read Pair	11.80 µs	1180 cycles
Complete Test (20 transactions)	123.40 µs	12,340 cycles

10. Integration Guidelines

10.1 Hardware Requirements

10.1.1 Pull-up Resistors

I2C requires external pull-up resistors on both SCL and SDA lines. The implementation uses tri-state outputs that only drive low, relying on pull-ups to achieve the high state.

- **Typical Values:** 2.2 kΩ - 10 kΩ depending on bus capacitance
- **Power Supply:** Connect to VDD (3.3V or 5V)
- **Location:** One set of resistors per bus (not per device)

10.1.2 Level Shifters

If connecting devices with different voltage levels, use bidirectional level shifters designed for I2C. Standard logic level shifters may not work due to the open-drain nature of I2C.

10.2 Software Integration

10.2.1 Initialization Sequence

17. Assert reset (rst_n = 0) for at least 10 clock cycles
18. Release reset (rst_n = 1)
19. Wait for 10 clock cycles for internal state stabilization
20. Begin normal operation

10.2.2 Write Transaction Procedure

21. Wait for busy = 0 (bus idle)
22. Set slave_address to target device address
23. Set rw = 0 (write operation)
24. Set data_in to 12-bit value to write
25. Pulse start = 1 for one clock cycle
26. Wait for busy = 0 (transaction complete)
27. Check ack_error flag (0 = success, 1 = error)

10.2.3 Read Transaction Procedure

28. Wait for busy = 0 (bus idle)
29. Set slave_address to target device address
30. Set rw = 1 (read operation)
31. Pulse start = 1 for one clock cycle
32. Wait for busy = 0 (transaction complete)
33. Check ack_error flag
34. Read 12-bit data from data_out

10.3 Multi-Slave Configuration

Multiple slaves can share the same I2C bus. Each slave must have a unique address configured via the SLAVE_ADDR parameter. The master addresses each slave individually using the slave_address input.

- **Address Uniqueness:** Ensure no two slaves have the same address
- **Reserved Addresses:** Avoid 7'h00-7'h07 and 7'h78-7'h7F (I2C standard reserved)
- **Bus Loading:** More slaves increase capacitance; may need lower resistor values

11. Troubleshooting Guide

11.1 Common Issues

Symptom	Possible Cause	Solution
ACK error on all transactions	Pull-up resistors missing or incorrect	Add 2.2-10 kΩ pull-ups to SCL and SDA
ACK error on all transactions	Slave address mismatch	Verify slave_address matches SLAVE_ADDR parameter
No SCL clock generated	DIVIDER too small (<2)	Increase DIVIDER parameter
Incorrect read data	Insufficient settling time	Increase wait cycles after busy=0 in driver
Bus hangs/stuck	Clock stretching not supported	This implementation doesn't support clock stretching
Data corruption	Multiple masters on bus	This implementation is single-master only

12. Appendices

12.1 Acronyms and Abbreviations

Acronym	Definition
ACK	Acknowledge
DUT	Design Under Test
DXA	Drawing eXchange Algorithm units
FSM	Finite State Machine
I2C	Inter-Integrated Circuit
LSB	Least Significant Bit
MSB	Most Significant Bit
NACK	Not Acknowledge
OE	Output Enable
R/W	Read/Write
SCL	Serial Clock Line
SDA	Serial Data Line
UVM	Universal Verification Methodology