**I2C Controller Verification Environment Specifications and Overview**

Table of Contents

[Document history 3](#_Toc194061540)

[1.Introduction 4](#_Toc194061541)

[1.1 Scope 4](#_Toc194061542)

[1.2 Abbreviations 4](#_Toc194061543)

[1.3 References 4](#_Toc194061544)

[2. I2C Controller Overview 5](#_Toc194061545)

[2.1. I2C Controller Features 5](#_Toc194061546)

[2.1.1. Master mode features 5](#_Toc194061547)

[2.1.2. Slave mode features 5](#_Toc194061548)

[2.2. I2C Controller Interfaces 6](#_Toc194061549)

[3. Verification Environment 7](#_Toc194061550)

[3.1 Environment Overview 7](#_Toc194061551)

[3.2 Components description 8](#_Toc194061552)

[3.2.1 APB Agent 8](#_Toc194061553)

[3.2.2. I2C Agent 9](#_Toc194061554)

[3.3 Register model 11](#_Toc194061555)

[3.4 Checkers 12](#_Toc194061556)

[3.6 Tests 13](#_Toc194061557)

[4. Simulation flow 14](#_Toc194061558)

[5. How to run a test 15](#_Toc194061559)

[6. Limitations and assumptions 15](#_Toc194061560)

# Document history

| Version | Date | Approved by | Created by | Description |
| --- | --- | --- | --- | --- |
| 0.1 | 03/03/2025 | Dorian Stan | Zsolt Laszlo | Document created |
|  |  |  |  |  |

# 1.Introduction

## 1.1 Scope

This document describes the verification environment for the I2C\_Controller using UVM

## 1.2 Abbreviations

|  |  |
| --- | --- |
| **DUT** | Device Under Test |
| **APB** | Advanced Peripheral Bus |
| **I2C** | Inter-Integrated Circuit |

## 1.3 References

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Name** | **Revision** | **Description** |
| 1 | I2C\_Controller Specification.pdf | 1.0 | DUT RTL specification |
| 2 | I2C\_Controller Metric plan.xlsx | 1.2 | Coverage, Checkers and Test plans |

# 2. I2C Controller Overview

* 1. A diagram of a bridge

     AI-generated content may be incorrect.

Figure 1. The I2C\_controller input/output interfaces & architecture diagram

## 2.1. I2C Controller Features

* Master/Slave functionality
* Variable clock divider
* Variable transfer length up to 256 bytes
* Large data transfer with minimal software access
* Configurable interrupt request interface

### 2.1.1. Master mode features

* Clock stretching: Slave devices can slow down the transfer rate by driving the SCL low for the required period and then releasing it
* Clock synchronization and arbitration: When two master devices try to control SCL, the line will be held low by the device with the longest low period. The first master that will detect SDA being held low while trying to drive it high loses arbitration. Large data transfer with minimal software access

### 2.1.2. Slave mode features

* While in slave mode, the device will stop processing a WRITE operation when it detects a STOP condition. If the protocol is violated and the START/STOP condition is generated in the middle of a transfer, the data byte for that particular transfer is discarded and the transfer stopped
* SR ( Repeated Start ) is supported while in Slave mode but with certain limitations. Every SR condition is treated as a STOP and the previous transfer is ended, giving registers access from the APB side until the address following the SR condition is matched. This interval can be used either to process the data or stop future transfers by setting ENABLE\_ACK to 0.
* When in slave mode, SCL and SDA are sampled using func\_clock and as a requirement, its frequency must always be at least 6 times higher than the SCL frequency. This also applies to START and STOP conditions as the time difference between the SDA toggle and the SCL toggle needs to be at least equal to the HIGH period of the SCL.

## 2.2. I2C Controller Interfaces

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **interface** | **signal** | **witdh** | **direction** | **note** |
| **APB** | psel | 1 | I | active high, indicates start of APB transaction |
| penable | 1 | I | active high, device enable, indicates acces pahse of transaction |
| pwrite | 1 | I | APB read -> 0 | APB write -> 1 |
| paddr | 6 | I | address of register to be accessed |
| pwdata | 32 | I | data to be written in the regster |
| prdata | 32 | O | data read from the register |
| pready | 1 | O | slvave ready |
| pslverr | 1 | O | slave error |
| **I2C** | scl | 1 | I/O | Serieal clock |
| sda | 1 | I/O | Serial data |
| **INTERRUPT** | irq | 1 | O | interrup request |

# Verification Environment

## 3.1 Environment Overview

* 1. The I2C\_Controller verification environment diagram is presented in Figure 2.

1. A diagram of a software

   AI-generated content may be incorrect.

Figure 2. The I2C\_Controller Verification Environment Diagram

## 3.2 Components description

### 3.2.1 APB Agent

#### Transaction

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Type** | **Description** |
| kind | 1 | apb\_trans\_kind\_t | Transaction type [read/write] |
| addr | AW | bit | Address |
| data | DW | bit | Data |
| delay | 32 | bit | Inter transaction delay |
| delay\_kind | 3 | apb\_delay\_kind\_t | Ranges of delays |
| resp | 1 | apb\_trans\_resp\_t | Response kind [pslverr] |
| ready\_delay | 32 | bit | Intra transaction delay |

A diagram of a data flow

AI-generated content may be incorrect.A diagram of a data processing process

AI-generated content may be incorrect.

#### Sequencer

* Randomizes the transaction
* Sends the transaction to the driver via TLM port

#### Driver

* Receives transaction from sequencer

**Master:**

* + Drives PSEL high along with address, op type and data (case specific)
  + Wait a clock cycle and drive PENABLE high
  + Wait PREADY along with the response kind and the data (case specific)

**Slave:**

* Wait for PSEL to be driven high
* Get address and operation type
* Drive PREADY high with a specified delay along with the response kind and the data (case specific)

#### Monitor

* Count the number of clock cycles until a transaction arrives
* After PSEL is driven high get the operation type and data (case specific)
* Wait until the end of the transaction and get the response kind and data (case specific)
* Send the collected transaction to the subscribers and scoreboard via analysis port

#### Coverage

* Get the transaction from the monitor
* Sample the interface’s specific cover groups

### 3.2.2. I2C Agent

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Type** | **Description** |
| kind | 1 | i2c\_trans\_kind\_t | Transaction type [read/write] |
| addr | 7 | bit | Address |
| data\_q | 8 [$] | bit | Data queue |
| resp | 1 [$] | i2c\_resp\_kind\_t | Response kind [ack/nack] |
| repeated\_start | 1 | bit | Repeated start condition |
| clock\_streching | 1 | bit | Clock streaching |



#### Sequencer

* Randomizes the transaction
* Sends the transaction to the driver via TLM port

#### Driver

* Receives transaction from sequencer

**Master:**

* + Drive scl in a parallel thread
  + Drive address bit by bit after each negative edge of scl plus a delay
  + Drive the operation type (read/write) bit
  + Get ACK or NACK from slave
  + If NACK detected skip the data phase
  + Read → every 9th edge drive ACK or NACK from the response queue
  + Write → drive data from queue bit by bit and wait response
  + After a stop condition or a NACK get drive a stop condition if repeated start is 0

**Slave:**

* Wait for a start condition
* Get the address and the op type
* If address does not match the device’s drive NACK
* Read → drive data from queue bit by bit and wait response
* Write → every 9th edge drive ACK or NACK from the response queue
* Perform clock stretching if called for
* Stop driving when stop condition detected

#### Monitor

* Wait start condition
* Sample data every positive edge of scl
* Convert address and data from serial to parallel
* Store address and operation type
* Store data in a queue
* Store responses in a queue
* Check repeated start
* Send transaction to subscribers and scoreboard via analysis port after a stop condition

#### Coverage

* Get the transaction from the monitor
* Sample the interface’s specific cover groups

## 3.3 Register model

A diagram of a software system

AI-generated content may be incorrect.

Figure 3. Register model functionality

* All the legal read and write APB sequences are initiated via the register models adapter.

APB sequences which aim for PSLVERR to be asserted are launched outside of the register model.

* The reading sequence also includes and automated checking mechanism utilizing the mirror method.
* The read only registers are not updated in the model, the mirror value is predicted before reading.

## 3.4 Checkers

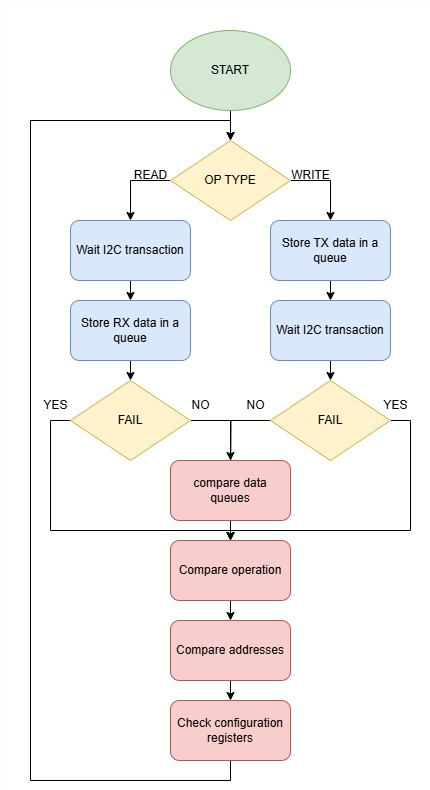


Figure 3. Main checking (located in scoreboard) flowchart

The protocol specific checkers are in their corresponding agent’s monitor or in the assertions located in the protocol’s interface file.

The main checking (device and registers) is in the scoreboard

All the checkers and coverage implementations can be found in the checkers/coverage sections of the attached metric plan.

## 3.6 Tests

The tests’ description can be found in I2C\_Controller metric driven plan.xlsx, the “Tests” sheet.

# 4. Simulation flow

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 4. Simulation flow of a standard test

# 

# 5. How to run a test

Running a scrip through Makefile:

* make run\_\* TESTNAME=<test\_name> SEED=<seed> VERBOSITY=<verbosity> => **for singular test**
* make run\_regression => **for running all tests**

# 6. Limitations and assumptions