**Verification Specifications**

**Date: 7 / 11 / 2023**

**Department: Semi-conductors**

**Your contact: paunchicia@gmail.com**

Revisions follow-up

|  |  |  |
| --- | --- | --- |
| Date | Reference no. | Reason for the modification |
| 16/06/2023 | 0.1 | Initial version – draft |
| 27/11/2023 | 0.2 | Final version |

Content

[1 Abbreviations and acronyms 4](#_Toc139045236)

[2 Introduction 5](#_Toc139045237)

[2.1 Purpose 5](#_Toc139045238)

[2.2 Scope 5](#_Toc139045239)

[2.3 Audience 5](#_Toc139045240)

[2.4 Applicable and reference documents 5](#_Toc139045241)

[3 System description 6](#_Toc139045242)

[4 Verification Strategy 10](#_Toc139045243)

[5 Verification Requirements 11](#_Toc139045244)

[6 Testbench and Test Environment 12](#_Toc139045245)

[7 Verification Methodologies 15](#_Toc139045246)

[8 Coverage Metrics 17](#_Toc139045247)

[9 Verification Deliverables 18](#_Toc139045248)

[10 Verification Schedule 19](#_Toc139045249)

1. Abbreviations and acronyms

The following list includes only the new abbreviations used in the document (compared to the abbreviations used in the specifications).

|  |  |
| --- | --- |
| IP  RTL  DV  REG  AMBA  APB  SOC | Intelectual Property  Register transfer level  Design Verification  Register (Flip-Flop)  Advanced Microcontroller Bus Architecture  Advanced Peripheral Bus (part of the AMBA)  System On Chip |

1. Introduction
   1. Purpose

The purpose of this verification document is to outline the process and methodology used to verify the functionality and performance of the APB protocol in a VLSI project.

* 1. Scope

The scope of this document includes a detailed description of the verification environment, test cases, and results.

* 1. Audience

The intended audience for this document includes design engineers, verification engineers, and system architects who are involved in the development and implementation of systems that use the APB protocol.

* 1. Applicable and reference documents

|  |  |  |
| --- | --- | --- |
| Document | ID | Revision / Date |
| AFVIP\_Specification\_Document | VS | 7.0/15th May 23 |

1. System description

1.1 Features

This module implements an arithmetic unit configurable and controllable through APB interface. Supports only Addition and Multiplication operations.

Fig. 1 ( Block diagram)

1. Single clock domain (1GHz)
2. Support asynchronous reset active low
3. The IP uses APB with addresses on 16 bits and data on 32 bits for configuration and status access. Please see the address map for all configuration registers:

|  |  |  |  |
| --- | --- | --- | --- |
| **APB Address** | **Type** | **Name** |  |
| **0x0000 – 007C** | RW | Instruction registers | Read/Write access |
| **0x0080** | RW | Configuration instruction | Read/Write access |
| **0x0084** | RO | Interrupt status | Read Only |
| **0x000C** | RWA | Interrupt control | Read/Write Auto reset |
| **0x0010** | RWA | Control Register | Read/Write Auto reset |

4. Instruction format:



* Imm – Immediate value
* DST – Destination register address
* RS0 – Source register 0
* RS1 – Source register 1
* Opcode – Operation code

5. The module supports the next operations according to the operation code:

* + Opcode == 3’d0: reg[dst] = reg[rs0] + imm
  + Opcode == 3’d1: reg[dst] = reg[rs0] \* imm
  + Opcode == 3’d2: reg[dst] = reg[rs0] + reg[rs1]
  + Opcode == 3’d3: reg[dst] = reg[rs0] \* reg[rs1]
  + Opcode == 3’d4: reg[dst] = reg[rs0] \* reg[rs1] + imm

6. If the arithmetic operation result exceeds 32 bits, the result written in the destination register will be overlapped: for example

* + 0xFFFFFFFE + 0x2 = 0x1
  + 0xFFFFFFFF \* 0x2 = 0xFFFF FFFE

7. Include a set of 32 RW registers mapped in next way:

|  |  |  |
| --- | --- | --- |
| **APB Address** | **RS0/RS1/DST address** | **Name** |
| **0x0000** | 0x00 | reg[00] |
| **0x0004** | 0x01 | reg[01] |
| **0x0008** | 0x02 | reg[02] |
| **0x000C** | 0x03 | reg[03] |
| **0x0010** | 0x04 | reg[04] |
| **0x0014** | 0x05 | reg[05] |
| **0x0018** | 0x06 | reg[06] |
| **0x001C** | 0x07 | reg[07] |
| **0x0020** | 0x08 | reg[08] |
| **0x0024** | 0x09 | reg[09] |
| **0x0028** | 0x0A | reg[10] |
| **0x002C** | 0x0B | reg[11] |
| **0x0030** | 0x0C | reg[12] |
| **0x0034** | 0x0D | reg[13] |
| **0x0038** | 0x0E | reg[14] |
| **0x003C** | 0x0F | reg[15] |
| **0x0040** | 0x10 | reg[16] |
| **0x0044** | 0x11 | reg[17] |
| **0x0048** | 0x12 | reg[18] |
| **0x004C** | 0x13 | reg[19] |
| **0x0050** | 0x14 | reg[20] |
| **0x0054** | 0x15 | reg[21] |
| **0x0058** | 0x16 | reg[22] |
| **0x005C** | 0x17 | reg[23] |
| **0x0060** | 0x18 | reg[24] |
| **0x0064** | 0x19 | reg[25] |
| **0x0068** | 0x1A | reg[26] |
| **0x006C** | 0x1B | reg[27] |
| **0x0070** | 0x1C | reg[28] |
| **0x0074** | 0x1D | reg[29] |
| **0x0078** | 0x1E | reg[30] |
| **0x007C** | 0x1F | Reg[31] |

8. The HW-SW handshake will be done according to the control and status registers and the interrupt:

* afvip\_intr – is a level output signal and can be triggered for 2 reasons:
* The module finished the instruction execution
* The module is wrong configured (unsupported opcode)
* ev\_ctrl\_start – Is an event type register and is controlled through APB. When this register is written through APB with 1, the module will start processing the configured instruction.
* sts\_intr\_error – Is a status register that can be read-only through APB. It indicates when the interrupt is raised because of an illegal configuration.
* sts\_intr\_finish – Is a status register that can be read-only through APB. It indicates when the interrupt is raised because the instruction execution is finished.
* ev\_intr\_clr\_err – It is an event type register, and its job is to clear the error interrupt.
* ev\_intr\_clr\_finish – It is an event type register, and its job is to clear the finish interrupt.

9. The HW-SW handshake for 1 instruction execution is done in 5 steps:

* Step 1 - Configure Registers through APB (Instruction, set values)
* Step 2 - Set start register through APB
* Step 3 - Wait for interrupt
* Step 4 - Read interrupt status
* Step 5 - Clear interrupt

10. The Interrupt must be raised in minimum 10 cycles from APB transfer completion of event\_control\_start register write-access with “1” value.

1.2 Limitations

The configuration must be stable during the instruction execution, From start event until to the interrupt indication.

1.3 Limitations

|  |  |
| --- | --- |
| **Parameter name** | **Description** |
| **TP** | Propagation time for simulation |

1.4 Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Size** | **Description** |
| **System Interface** | | | |
| **clk** | I | 1 | Clock |
| **rst\_n** | I | 1 | Asynchronous Reset active low |
| **afvip\_intr** | O | 1 | Interrupt |
| **APB Interface** | | | |
| **psel** | I | 1 | Select. |
| **penable** | I | 1 | Enable. |
| **paddr** | I | 16 | Address. |
| **pwrite** | I | 1 | Direction. |
| **pwdata** | I | 32 | Write data. |
| **pready** | O | 1 | Ready. |
| **prdata** | O | 32 | Read data. |
| **pslverr** | O | 1 | Transfer error. |

1. Verification Strategy

**Functional Tests**

Functional tests will verify that this module performs its intended functions correctly. This will include testing the arithmetic operations, the behavior of the APB interface, and the handling of unsupported opcodes.

For example, a test can be created that sets the operation code to 3’d0, sets values in the rs0 and imm registers, and checks that after a few clock cycles, the value in the dst register is the sum of the values in the rs0 and imm registers.

1. Verification Requirements

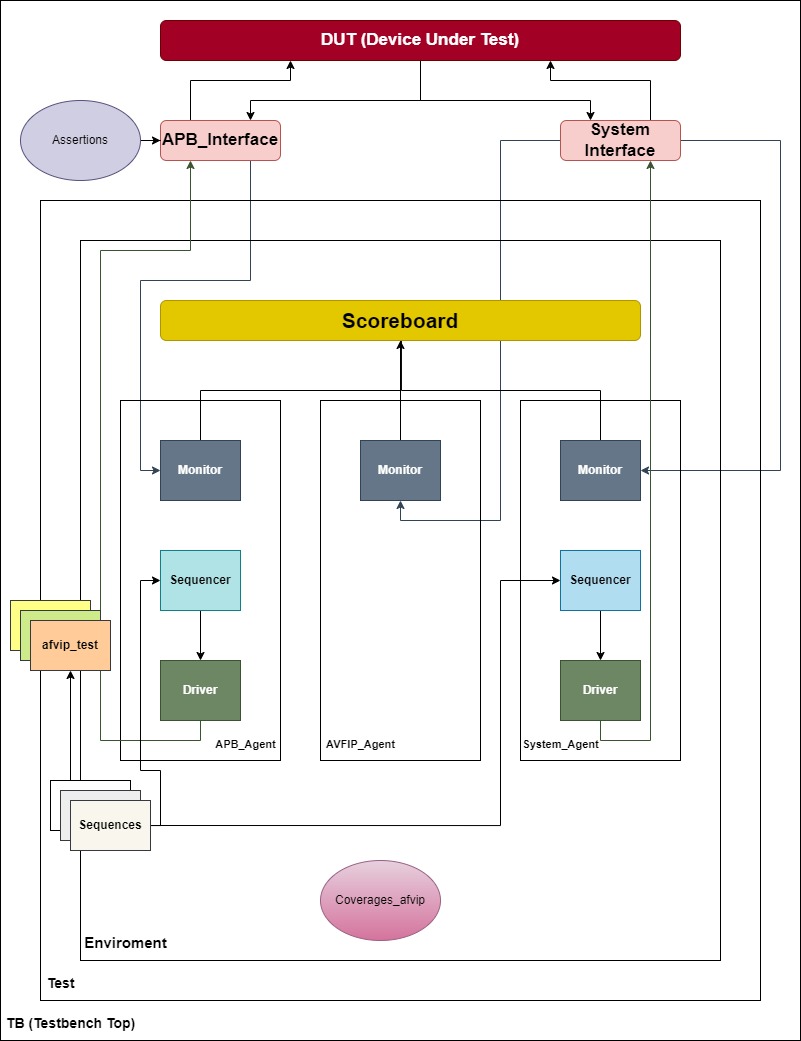
**Functional Requirements**

* The module should correctly perform addition and multiplication operations according to the operation code and the values in the rs0, rs1, and imm registers.
* The module should correctly handle unsupported opcodes by triggering the afvip\_intr signal.
* The module should correctly implement the APB interface, including the behavior of the psel, penable, paddr, pwrite, pwdata, pready, prdata, and pslverr signals.  
  The module should correctly handle asynchronous reset active low through the rst\_n signal.

**Non-Functional Requirements**

* The module should meet its performance requirements, such as its maximum operating frequency of 1GHz and its latency between setting an operation code and seeing the result in the dst register.
* The module should be able to handle high levels of stress, such as high data rates or high levels of concurrency.

1. Testbench and Test Environment

****

**DUT (Device Under Test)**

This is the module that is being tested, which in this case is the arithmetic unit configurable and controllable through an APB interface.

**APB\_Interface**

This component provides an interface to the APB bus, allowing the testbench to communicate with the DUT. **System\_Interface**

This component provides an interface to the system-level signals, such as clock and reset. **Scoreboard**

This component checks the results of the DUT against expected values to ensure that it is functioning correctly. It could be used to verify the arithmetic unit module by monitoring the APB interface to capture inputs and outputs, calculating the expected result of operations, and comparing it with the actual result produced by the DUT. The scoreboard could also check for other conditions specific to this module, such as the triggering of the afvip\_intr signal and the handling of unsupported opcodes. If any discrepancies are found, the scoreboard would raise an error. **Monitor**

This component observes the signals on the DUT and reports them to the scoreboard for checking. **Sequencer**

This component generates stimulus for the DUT by sending sequences of transactions to the driver. **Sequences**

These are pre-defined sets of transactions that are sent to the driver by the sequencer. **Driver**

This component translates the transactions from the sequencer into signal-level activity on the DUT.

**Hardware Requirements**

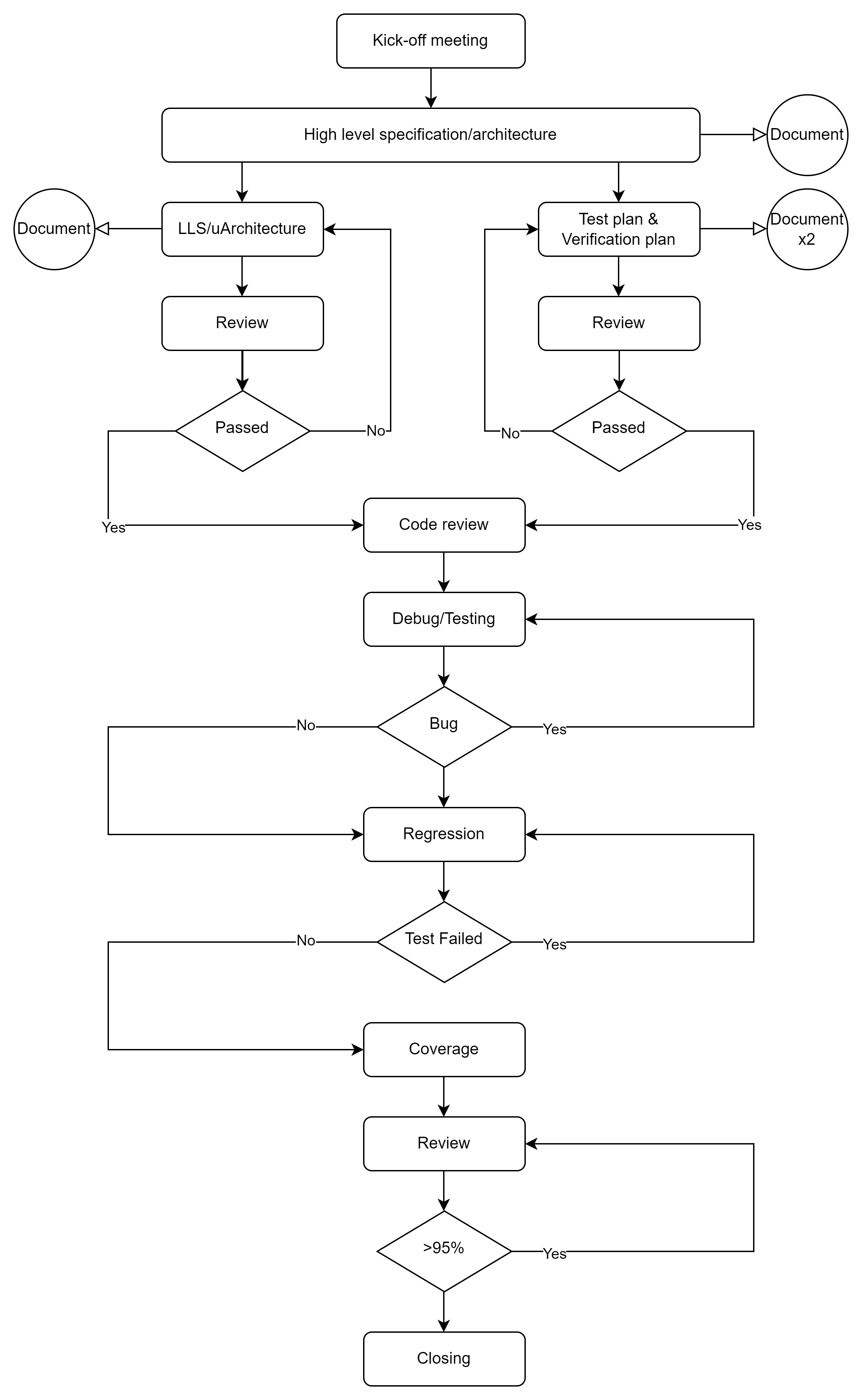
A computer or server with sufficient processing power and memory to run simulations, formal verification, and emulation tools.  
If emulation is used, an emulation platform that supports the design of this module and its interfaces.

**Software Requirements**

Simulation tools, such as Vivado, to run simulation tests on the design.  
Formal verification tools to mathematically prove that the design meets its specifications.  
Emulation tools to run tests on the design at a higher speed than simulation.  
A testbench framework to manage the execution of tests and the collection of results.  
Network Requirements  
If emulation is used, a network connection may be required to connect the emulation platform to other components in a system-level test environment.

**Test Setup**

A testbench should be created that instantiates this module and provides stimuli to its inputs (clk, rst\_n, psel, penable, paddr, pwrite, and pwdata) and checks its outputs (pready, prdata, and pslverr).  
Test scenarios should be created that cover the functional requirements of this module, such as verifying that it correctly performs arithmetic operations, handles unsupported opcodes, and implements the APB interface.  
Test stimuli should be created that provide appropriate input values to this module for each test scenario.  
Any necessary test data, such as values for the rs0, rs1, and imm registers or expected results for arithmetic operations, should be prepared.

1. Verification Methodologies

**Kick-off meeting**

A discussion of the project's objectives, parameters, and schedule.

**High level specification/architecture**  
The production of a high-level design that describes the architecture and requirements of the system.

**LLS/uArchitecture & Testplan verification**

The development of a low-level specification or microarchitecture and the verification of the test plan.

**Code review**

A review of the code to ensure it meets the project’s standards and requirements.

**Debug/Testing**

The process of finding and fixing bugs in the code through testing.

**Regression**

The process of running tests to ensure that changes to the code do not introduce new bugs.

**Coverage**

The measurement of how much of the code is being tested.

**Review (must be above 95%)**

A review of the project to ensure that it meets the required coverage threshold of 95% or above.

**Closing**

The final step in the project where all tasks are completed and the project is closed.

1. Coverage Metrics

For the APB module, several coverage metrics can be used to measure the completeness of the verification process. These include statement coverage, branch coverage, and functional coverage. This document describes these coverage metrics and outlines the goals for achieving high levels of coverage during the verification of the APB module.

**Statement and brench coverage**

Statement coverage measures the percentage of code statements in the DUT that have been executed during simulation. The goal is to achieve 100% statement coverage, meaning that every line of code in the DUT has been executed at least once.  
  
Branch coverage measures the percentage of branches or decision points in the DUT that have been executed during simulation. The goal is to achieve 100% branch coverage, meaning that every possible outcome of each decision point in the DUT has been exercised.  
  
To achieve high levels of statement and branch coverage for the APB module, test cases should be created that exercise all possible opcodes and combinations of registers and immediate values. This will ensure that all lines of code and decision points in the DUT are exercised during simulation.  
  
**Functional Coverage**

Functional coverage measures the percentage of specified functionality in the DUT that has been verified during simulation. The goal is to achieve 100% functional coverage, meaning that all specified functionality in the DUT has been exercised and verified.  
  
For the APB module, functional coverage can be achieved by defining coverage points for each supported opcode and ensuring that all possible combinations of registers and immediate values have been exercised for each opcode.  
  
For example, functional coverage points could be defined for each of the following operations:  
  
Opcode == 3’d0: reg[dst] = reg[rs0] + imm  
Opcode == 3’d1: reg[dst] = reg[rs0] \* imm  
Opcode == 3’d2: reg[dst] = reg[rs0] + reg[rs1]  
Opcode == 3’d3: reg[dst] = reg[rs0] \* reg[rs1]  
Opcode == 3’d4: reg[dst] = reg[rs0] \* reg[rs1] + imm  
The goal would be to achieve 100% functional coverage, meaning that all possible combinations of registers and immediate values have been exercised for each opcode.

1. Verification Deliverables

The verification of the APB module involves the generation of various documentation and reports, including revision follow-up, project description, test plan, coverage, assertion .  
  
**Accessing Verification Information**

**In order to view the test results, coverage reports, and other verification information for the APB module, it is necessary to access a specific document named Test\_plan.xlsx**

1. Verification Schedule

