Maseeh College of Engineering
and Computer Science
PORTLAND STATE UNIVERSITY

ECE 593: SUMMER 2023 - CLASS PROJECT
FUNDAMENTALS OF PRE-SILICON VALIDATION

VERIFICATION OF AN AHB2APB BRIDGE

CHECKPOINT 2
VERIFICATION PLAN

MOHAMED GHONIM
GAYATRI VEMURI

07/17/2023

Table of Contents

Introduction and Outline

The purpose of this Verification Plan is to outline the approach and methodology we will use to verify the functionality and performance of the AHB-APB bridge design. The AHB-APB bridge is a critical component that is responsible for facilitating communication between the high-performance system bus (AHB) and the low-power peripheral bus (APB). Given its pivotal role, it is essential that the bridge operates correctly under all possible conditions and scenarios.

Our verification plan is designed to be comprehensive and rigorous, covering different aspects of the bridge's functionality. We will be using Questasim to run the simulation and SystemVerilog (SV) and SystemVerilog Assertions (SVA) to write the testbench. Our plan is divided into two phases: the first phase involves creating a SystemVerilog testbench, due by 7/31/2023, and the second phase involves creating a Universal Verification Methodology (UVM) testbench, due by 8/17/2023.

The verification Plan for the AHB-APB Bridge Design

The verification plan consists of two main requirements, each consisting of sub-requirements:

1- Technical requirements
   a. Verification Levels
   b. Functions
   c. Specific Tests & Methods
      i. Type of verification
      ii. Verification Strategy
      iii. Abstraction level
      iv. Checking
   d. Coverage
   e. Scenarios

2- Project Management needs
   a. Tools
   b. Risks/Dependencies
   c. Resources
   d. Schedule

First, we will start with the high-level verification plan. This is a concise outline of our plan. It’s then followed by our low-level, more detailed verification plan.

High- Level Verification Plan

1) Technical Requirements

a) Verification Levels

The verification will be conducted at two levels: The block (module) level and the top level.

- At the block level, each component/module (AHB Master, AHB Slave Interface, APB FSM Controller, and APB Interface) will be verified individually.
- At the top level, the integrated AHB-APB Bridge will be verified as a whole.

b) Functions.

The functions to be verified are implemented in the AHB Master, AHB Slave Interface, APB FSM Controller, and APB Interface modules, however, we will refer to the AMBA specification document and extract the functions/functionalities of the bridge design and test the RTL Design we have. Each function will be tested under various conditions, including normal operation, edge cases, and error conditions.

- Address Latching. (Will verify)
- Address Decoding and Peripheral Select. (Will verify)
- Bus Handshake and Control Signal Retiming. (Will verify)
- Data Transfer. (Will verify)
- Reset and Clock Signal Handling. (Will verify)
- Error Handling. (Will verify)
- Timing Parameters. (Will verify)
- Support for Different Transfer Types. (Will verify)
- Support for Different Bus Masters. (Will verify)
- Support for Different Peripheral Devices. (Will verify)
- System Debug (Won't Verify):

c) Specific Tests and Methods: Environment.

- **Type of Verification: (Gray box)** Initially black box verification will be used as we are mostly interested in the input and output of the system, but we will also consider gray box verification since we have the design code and we plan to use some assertions in the controller's finite state machine for example, and it gray box would give us better controllability and observability especially in debugging.
- Verification Strategy: The strategy will involve driving inputs and checking outputs for each component. The verification will be deterministic, and we plan to use constraint random stimulus as well.
- Abstraction Level: The verification will be conducted at the intent/context level, it will be **transaction based** since this is most suitable for bus and bridge designs in our opinion.
- Checking: We will use transaction-based reference model checking.

d) Coverage requirements.

The target is to cover all functional stimulus requirements and to ensure the design has reached interesting points as well as corner cases. Coverage will be measured using code coverage tool in Questasim to ensure all lines of code and branches have been executed. We will also implement a

transaction model checking and coverage to make sure we covered all the interesting bus transactions.

e) Test Case Scenarios.

The test case matrix for the AHB-APB bridge design is a comprehensive set of tests designed to verify the functionality of the bridge under various conditions. The matrix is divided into 30 test cases, each with a unique set of inputs and expected outputs, with an additional 30 test cases focusing on reset signals and corner cases.

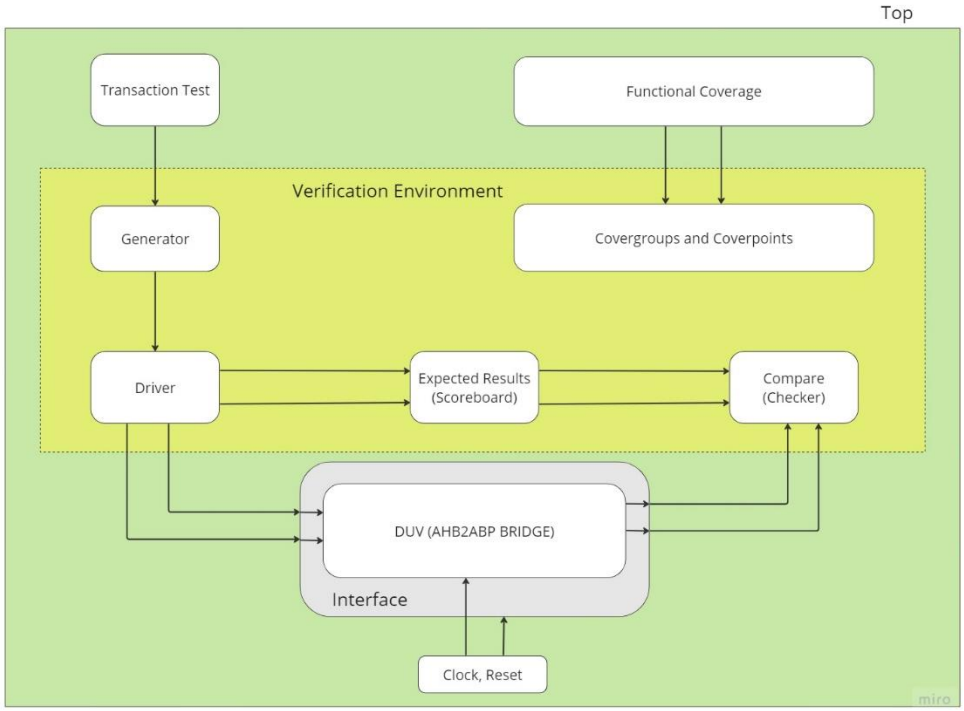
The test cases cover a range of scenarios, including different types of transactions (read, write), different sizes of data transfers (byte, halfword, word), and different burst types (single transfer, incrementing burst of unspecified length, 4-beat wrapping burst, 4-beat-incrementing burst, 8-beat wrapping burst, 8-beat incrementing burst).

The matrix also includes tests for corner cases, such as simultaneous read and write operations, and tests for the handling of error conditions, such as an invalid address or data size.

In addition, the matrix includes tests for the reset functionality of the bridge, ensuring that the bridge can correctly initialize and reset its state.

Each test case in the matrix is designed to verify a specific function or feature of the bridge, and the expected outputs for each test case are based on the specifications of the AHB-APB bridge.

The test case matrix provides a systematic and thorough approach to verifying the functionality of the AHB-APB bridge, ensuring that the bridge operates correctly under a wide range of conditions and scenarios.



2) Project Management Needs

a) Required tools.

- Questasim simulator will be used for simulation and coverage analysis.
- and SV and SVA will be used for writing the testbench.

b) Risks and Dependencies.

- Risks include potential bugs in the design and delays in the verification process given the first tight schedule.
- Dependencies include the availability of the Questasim simulator and license.

c) Resources.

The resources include:

- Two graduate students
- computing resources
- and software licenses.

d) Schedule.

- The SV testbench will be delivered on 7/31/2023, and the UVM testbench is due on 8/17/2023.

Low- Level Verification Plan

1) Technical Requirements

a) Verification Levels

1- Complexity of individual components:
   a. **AHB_slave_interface:** This module handles the interface with the AHB bus. It implements pipeline logic for address, data, and control signals. It also generates valid logic and implements tempselx logic. The complexity of this module is moderate due to the pipelining and logic generation involved.
   b. **APB_FSM_Controller:** This module is a finite state machine (FSM) that controls the APB bus. It has several states and transitions between them based on various conditions. The complexity of this module is high due to the FSM implementation.
   c. **APB_Interface:** This module interfaces with the APB bus. It maps input signals to output signals and generates random data for read operations. The complexity of this module is low.
   d. **Bridge_Top:** This module is the top-level module that instantiates the AHB_slave_interface and APB_FSM_Controller. The complexity of this module is low as it primarily serves as a structural component to connect the other modules.

2- Existence of clean interface and specification:
   a. The design has clean interfaces between the AHB bus, APB bus, and the bridge. Each module has clearly defined inputs and outputs, and the data flow between modules is well-structured.
   b. The specification of the design is clear, with each module's functionality and interaction with other modules well-defined. The FSM in the APB_FSM_Controller is particularly well-specified, with clear states and transitions.

We also have access to the AMBA specification document from ARM. Based on these factors, it would be beneficial to verify the individual components independently due to their differing complexities. This would allow for more focused and efficient testing. After verifying the individual components, the entire design can be verified as a whole to ensure the correct interaction between components.

b) Functions.

1- Tag all functions as will/won't verify:

a. **Address Latching (will verify):**
The bridge latches the address and holds it valid throughout the transfer. This ensures that the correct address is used for the entire duration of the transfer.

b. **Address Decoding and Peripheral Select (will verify):**
The bridge decodes the address and generates a peripheral select signal. Only one select signal can be active during a transfer. This ensures that the correct peripheral is selected for the transfer.

c. **Bus Handshake and Control Signal Retiming (will verify):**
The bridge handles the bus handshake and control signal retiming on behalf of the local peripheral bus. This ensures that the bridge can correctly interface with both the AHB and the APB.

d. **Data Transfer (will verify):**
The bridge is responsible for transferring data between the AHB and the APB. This includes both read and write operations.

e. **Reset and Clock Signal Handling (will verify):**
The bridge handles the reset and clock signals. This ensures that the bridge operates correctly in the overall system context.

f. **Error Handling (will verify):**
The bridge should handle any errors that occur during the transfer process. This includes both detecting errors and responding appropriately.

g. **Timing Parameters (will verify):**
The bridge must adhere to the timing parameters specified in the AMBA specification. This includes parameters for input signals and output signals.

h. **Support for Different Transfer Types (will verify):**
The bridge should support different types of transfers, including single transfers and burst transfers.

i. **Support for Different Bus Masters (will verify):**
The bridge should correctly handle transfers initiated by different bus masters.

j. **Support for Different Peripheral Devices (will verify):**
The bridge should correctly interface with a variety of peripheral devices on the APB.

k. **System Debug (Won't Verify):**
The RTL code does not explicitly mention system debug functionality. However, system debugging is often a pervasive function that is integrated into the design of the system at a higher level.

2- Define conditions under which each function will be checked:

a. **System Reset:** The system reset function should be checked at the start of the simulation to ensure that the system starts in a well-known state.

b. **Data Write:** The data write function should be checked under conditions were valid data is being written to the system.

c. **Data Read:** The data read function should be checked under conditions were valid data is being read from the system.

d. **Addressing:** The addressing function should be checked under conditions where valid addresses are being accessed.

3- Establish and review verification priority order:

a. Critical: System reset, data write, data read, and addressing are all critical functions that need to be correct for other checking to proceed.
b. Not verified at this level: System debug is not explicitly mentioned in the RTL code and may not be verified at this level.
c. Secondary: Once the critical functions have been verified, additional checks can be performed to ensure that the system behaves correctly under a variety of conditions. This might include checking the system's response to invalid data, invalid addresses, and other error conditions.

c) Specific Tests and Methods: Environment.

1- Type of Verification

a. Black box, white box, or grey box? (Gray box)

i. In our initial stages, we will employ black box verification as we are primarily interested in the input and output of the system. This approach is suitable for our preliminary testing where we aim to verify if the system behaves as expected for given inputs. We also plan to use assertions to perform sanity check and to verify finite state machine transitions.

ii. As we progress, we will transition to grey box verification. This hybrid approach leverages our knowledge of the internal workings of the system. This strategy is particularly beneficial for our AHB-APB bridge design as it enables us to gain a deeper understanding of the interaction between different components (AHB Master, AHB Slave Interface, APB FSM Controller, and APB Interface) and how they contribute to the overall functionality of the system.

b. Dependencies:

i. **Functions to be verified:** The functions we plan to verify include system reset, data write, data read, and addressing. These functions are critical for the operation of the AHB-APB bridge.
ii. **How to best exercise internal structures:** For the grey box verification, we will create specific test cases that exercise the internal structures of the system. For instance, we can create test cases that specifically target the transitions between different states in the APB_FSM_Controller.
iii. **How errors manifest themselves:** Errors can manifest themselves in various ways such as incorrect data being read/written, the system not transitioning to the correct

state, etc. By understanding the internal workings of the system, we can better identify and diagnose these errors.

iv. **Resources for maintenance:** The resources for maintenance include us, the two graduate students, the Questasim simulator for running the simulations, and the SV and SVA for writing the testbench.

c. Maintenance cost vs observability benefit:

i. While grey box verification requires more effort and resources (maintenance cost) compared to black box verification, it provides better observability into the system. This can help in identifying and diagnosing issues more effectively, leading to a more robust and reliable system. Therefore, we believe that the observability benefit of grey box verification can justify the increased maintenance cost.

2- Verification Strategy

a. Describe how each component will drive inputs and check outputs:

i. **AHB_slave_interface:** We will drive inputs such as the AHB bus signals and check outputs such as the valid signal and the tempselx logic. We will also check the pipelining of address, data, and control signals.

ii. **APB_FSM_Controller:** We will drive inputs such as the valid signal, the Hwrite signal, and the Hwritereg signal, and check the state transitions of the FSM. We will also check the output signals that control the APB bus.

iii. **APB_Interface:** We will drive inputs such as the Pwrite signal and the Penable signal, and check outputs such as the Prdata signal and the Pready signal.

iv. **Bridge_Top:** We will drive inputs to the AHB_slave_interface and the APB_FSM_Controller, and check the outputs from these modules to ensure correct interaction.

b. Carefully consider reuse at multiple levels/domains:

i. We will design our test bench in a modular way to allow for reuse of test sequences and scenarios. This will enable us to efficiently test different components and different levels of the design.

c. Deterministic/random/formal:

i. **Deterministic:** This approach is suitable for small, simple, straightforward DUV. However, given the complexity of our design, a purely deterministic approach may not be sufficient.

ii. **Random:** Random testing can help uncover unexpected issues, but too much random testing could miss problems due to lack of focus, and too little could miss problems due to insufficient coverage. We will use constrained random testing, which allows us to control the ability to inject sequence into the random environment. This approach is particularly suitable for our design as it has many permutations.

iii. **Formal Verification:** Formal verification is a good candidate for designs with many permutations. It uses mathematical methods to prove the correctness of a design under all possible conditions. However, it requires a significant amount of resources and expertise. Given our resources and timeline, we will focus on simulation-based verification (black box and grey box) for now, but we may consider formal verification for critical parts of the design if necessary.

3- Abstraction level

a- Work at intent/context level (higher than pin level):

1. We will work at **the transaction level**, which can be through of as the intent/context level, which is higher than the pin level. This means that we will focus on the behavior and functionality of the design, rather than the individual signals. This approach allows us to cover all interesting scenarios and provides a more comprehensive view of the system's behavior.

2. The choice of abstraction level may parallel the verification level. For instance, when we are doing black box testing, we are primarily interested in the input and output of the system, which aligns with the intent/context level. When we transition to grey box testing, we will still maintain this level of abstraction, but we will also have visibility into the internal workings of the system.

b. Target correct abstraction:

a. By working at the intent/context level, we can define concise test scenarios on interesting sequences. For example, we can create a test scenario that simulates a specific sequence of read and write operations to the AHB-APB bridge.
b. This level of abstraction also allows us to obtain meaningful reports. The reports will provide information about the behavior and functionality of the system, rather than just the state of individual signals. This can help us to better understand the system's behavior and identify any issues.

4- Checking

a. Based on box, stimulus strategy, abstraction level: Our checking strategy will be based on our grey box verification approach, our constrained random stimulus strategy, and our intent/context abstraction level. This means we will be checking both the external behavior of the system (inputs and outputs) and some aspects of the internal workings of the system, using a variety of test scenarios generated through constrained random stimulus.
b. **Check all outputs:** We will check all outputs of the system to ensure they behave as expected for the given inputs. This includes checking the state transitions of the FSM, the data read from the system, and the signals controlling the APB bus.
c. **Design context vs microarch vs arch checking:** Our checking will primarily be at the design context and microarchitecture levels, as we are focusing on the behavior and functionality of the AHB-APB bridge design and its components. We will check that the design behaves as expected in the context of its intended use, and that the microarchitecture (the internal workings of the design) operates correctly.
d. **Style: golden vector, reference model, transaction based:** Given our schedule and limited resources, as well as the availability of the RTL Design and the bridge documentation/user manual from ARM, the most feasible approach for us would be the Transaction Based Reference Model styles. Here's why:
    - **Transaction Based Reference Model:** This approach is feasible with our current resources. We can define a series of transactions (such as a read operation, a write operation, a reset operation, etc.) based on the bridge documentation/user manual from ARM. Then, we can check that our system behaves correctly for each of these transactions. This approach is particularly suitable for our grey box testing where we have some visibility into the internal workings of the system.

The Reference Model and Golden vector approaches, on the other hand, might be challenging given our current resources. Creating a cycle-accurate model of the AHB-APB bridge would require a significant amount of time and effort. Moreover, without a known-correct reference model, it would be difficult to ensure the accuracy of our own model. Therefore, given our schedule and limited resources, we suggest focusing on the Golden Vector and Transaction Based Reference Model styles for our verification plan.

d) Coverage requirements.

Here's how we can address the coverage requirements for the AHB-APB bridge design:

1- *Feedback/quality control mechanism:* We will use coverage metrics to provide feedback on the quality of our verification. These metrics will include functional coverage, code coverage, and assertion coverage. Functional coverage will tell us how much of the design's functionality has been exercised by our tests. Code coverage will tell us how much of the design's code has been exercised. Assertion coverage will tell us how many of our assertions have been proven or disproven. We will use these metrics to identify areas of the design that need more testing and to guide our test development.

2- *Target: cover all functional stimulus requirements:*

- **All types of commands:** We will ensure that our tests cover all types of commands that the AHB-APB bridge can handle, such as read and write commands.
- **Specific or varying range of data types:** We will ensure that our tests cover a range of data types that the AHB-APB bridge can handle. This includes different sizes of data and different patterns of data.
- **Legal concurrent stimulus:** We will ensure that our tests cover scenarios where multiple stimuli are applied concurrently. This includes scenarios where multiple commands are issued concurrently.
- **Initiator/responder drive errors into DUV:** We will ensure that our tests cover scenarios where the initiator or responder drives errors into the Design Under Verification (DUV). This includes scenarios where invalid commands are issued or where the response to a command is incorrect.

3- *Target: cover design has reached interesting points:*

- **Architecture, microarchitecture, design states and modes:** We will ensure that our tests cover all states and modes of the design. This includes all states of the APB_FSM_Controller and all modes of operation of the AHB-APB bridge.
- **Sequences of functional stimulus:** We will ensure that our tests cover different sequences of functional stimulus. This includes sequences of read and write operations, sequences of state transitions, and sequences of mode changes.
- **All possible output types:** We will ensure that our tests cover all possible output types of the design. This includes all possible values of the output signals and all possible states of the output registers.

4- *Coverage feedback also needed on Validation Environment:* We will also collect coverage data on our validation environment. This includes coverage of the test sequences that we have defined and coverage of the assertions that we have included in our testbench. This data will help us to identify areas of our validation environment that need improvement.

e) Test Case Scenarios.

a. *Characterize each test:*
   i. Each test will be characterized with a label, a description, and the targeted feature. For example, a test might be labeled "Read Operation Test", described as "This test checks the functionality of read operations", and targets the read operation feature of the AHB-APB bridge.
   ii. Each test will be cross-referenced to the functional requirements and coverage goals that it addresses. This will help us to ensure that all functional requirements and coverage goals are covered by our tests.

b. *Group tests with similar verification features into scenarios:*
   i. Tests with similar verification features will be grouped into scenarios. For example, all tests that check read operations might be grouped into a "Read Operations Scenario".
   ii. Each scenario will be characterized by its configuration, granularity, and strategy. The configuration describes the setup of the system for the scenario, the granularity

describes the level of detail of the tests in the scenario, and the strategy describes the approach taken to verify the features in the scenario.

c. Start with a basic set of tests and incrementally build:

i. We will start with a basic set of tests that cover the main features of the AHB-APB bridge. For example, we might start with tests for basic read and write operations.

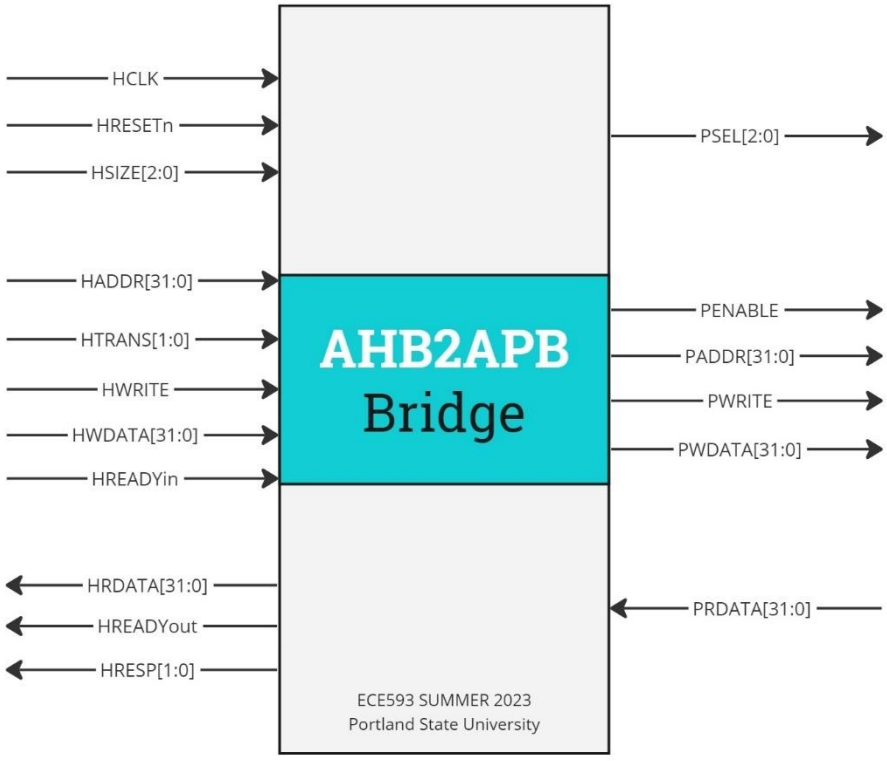
ii. We will then incrementally build on this basic set of tests, adding more tests to cover more features and more scenarios. This approach allows us to start testing quickly while still ensuring comprehensive coverage.

d. Random coverage-based matrix:

i. We will use a random coverage-based matrix to guide our test development. This matrix will specify what inputs are constrained or unconstrained, what configurations are needed, what variations of data items are tested, what are the important attributes of data, what are the interesting sequences for all DUV inputs, what are the error conditions to be tested, and what are the corner cases to be tested.

ii. For example, the matrix might specify that the Hwrite signal is constrained to be either 0 or 1, that we need configurations where the AHB-APB bridge is in different states, that we test both small and large data values, that we test sequences of read and write operations, that we test conditions where invalid commands are issued, and that we test cases where the AHB-APB bridge is at the limits of its operating conditions.

Below is a diagram of the inputs and outputs of the AHB to APB Bridge design, this will be essential to our top-level verification.



Below are some of the signals of special interest in our stimulus generation as well as monitoring and checking our transaction-based model. These will be especially important when verifying the design modules.

AHB Master

HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.

HTRANS[1:0]	Type	Description
00	IDLE	Indicates that no data transfer is required. The IDLE transfer type is used when a bus master is granted the bus, but does not wish to perform a data transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer should be ignored by the slave.
01	BUSY	The BUSY transfer type allows bus masters to insert IDLE cycles in the middle of bursts of transfers. This transfer type indicates that the bus master is continuing with a burst of transfers, but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. The transfer should be ignored by the slave. Slaves must always provide a zero wait state OKAY response, in the same way that they respond to IDLE transfers.
10	NONSEQ	Indicates the first transfer of a burst or a single transfer. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of one and therefore the transfer type is NONSEQUENTIAL.
11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the size (in bytes). In the case of a wrapping burst the address of the transfer wraps at the address boundary equal to the size (in bytes) multiplied by the number of beats in the transfer (either 4, 8 or 16).

HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.

3.7.1 Transfer direction

When **HWRITE** is HIGH, this signal indicates a write transfer and the master will broadcast data on the write data bus, **HWDATA[31:0]**. When LOW a read transfer will be performed and the slave must generate the data on the read data bus **HRDATA[31:0]**.

3.7.2 Transfer size

HSIZE[2:0] indicates the size of the transfer, as shown in Table 3-3.

Table 3-3 Size encoding

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size	Description
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

The size is used in conjunction with the **HBURST[2:0]** signals to determine the address boundary for wrapping bursts.

HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.

Table 3-2 Burst signal encoding

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

AHB Slave Interface

HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

During a transfer the slave shows the status using the response signals, **HRESP[1:0]**:

OKAY The OKAY response is used to indicate that the transfer is progressing normally and when **HREADY** goes HIGH this shows the transfer has completed successfully.

ERROR The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.

RETRY and SPLIT Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer.

In normal operation a master is allowed to complete all the transfers in a particular burst before the arbiter grants another master access to the bus. However, in order to avoid excessive arbitration latencies it is possible for the arbiter to break up a burst and in such cases the master must re-arbitrate for the bus in order to complete the remaining transfers in the burst.

Table 3-5 Response encoding

HRESP[1]	HRESP[0]	Response	Description
0	0	OKAY	When **HREADY** is HIGH this shows the transfer has completed successfully. The OKAY response is also used for any additional cycles that are inserted, with **HREADY** LOW, prior to giving one of the three other responses.
0	1	ERROR	This response shows an error has occurred. The error condition should be signalled to the bus master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition.
1	0	RETRY	The RETRY response shows the transfer has not yet completed, so the bus master should retry the transfer. The master should continue to retry the transfer until it completes. A two-cycle RETRY response is required.
1	1	SPLIT	The transfer has not yet completed successfully. The bus master must retry the transfer when it is next granted access to the bus. The slave will request access to the bus on behalf of the master when the transfer can complete. A two-cycle SPLIT response is required.

Testcases Matrix

Below is part of our testcase matrix. The full testcase matrix has 60 entries and is attached as a separate spreadsheet.

Test case ID	HTRANS	HSIZE	HBURST	HRESP	OPERATION	Description	Inputs	Expected Outputs	Functions Being Tested
TC1	NONSEQ	Byte	Single	Okay	Read	Single byte read	HADDR, HWDATA	PRDATA, HREADY	Read Operation
TC2	NONSEQ	Halfword	Single	Okay	Write	Single halfword write	HADDR, HWDATA	HREADY	Write Operation
TC3	NONSEQ	Word	Single	Okay	Read	Single word read	HADDR, HWDATA	PRDATA, HREADY	Read Operation
TC4	NONSEQ	Byte	Single	Error	Write	Single byte write operation with error	HADDR, HWDATA	HREADY, HRESP = Error	Write Operation with Error
TC5	NONSEQ	Halfword	Incrementing Burst	Okay	Read	Incrementing burst read with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC6	NONSEQ	Word	Incrementing Burst	Okay	Write	Incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
TC7	NONSEQ	Byte	4-beat wrapping	Okay	Read	4-beat wrapping burst read operation with byte size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC8	NONSEQ	Halfword	4-beat wrapping	Okay	Write	4-beat wrapping burst write	HADDR, HWDATA	HREADY	Write Operation with Burst

						operation with halfword size			
TC9	NONSEQ	Word	4-beat wrapping	Okay	Read	4-beat wrapping burst read operation with word size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC10	NONSEQ	Byte	4-beat wrapping	Okay	Write	4-beat incrementing burst write operation with byte size	HADDR, HWDATA	HREADY	Write Operation with Burst
TC11	NONSEQ	Halfword	4-beat wrapping	Okay	Read	4-beat incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC12	NONSEQ	Word	4-beat wrapping	Okay	Write	4-beat incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
TC13	NONSEQ	Byte	8-beat wrapping	Okay	Read	8-beat wrapping burst read operation with byte size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC14	NONSEQ	Halfword	8-beat wrapping	Okay	Write	8-beat wrapping burst write operation with halfword size	HADDR, HWDATA	HREADY	Write Operation with Burst
TC15	NONSEQ	Word	8-beat wrapping	Okay	Read	8-beat wrapping burst read operation with word size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC16	NONSEQ	Byte	8-beat wrapping	Okay	Write	8-beat incrementing burst write operation with byte size	HADDR, HWDATA	HREADY	Write Operation with Burst
TC17	NONSEQ	Halfword	8-beat wrapping	Okay	Read	8-beat incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
TC18	NONSEQ	Word	8-beat wrapping	Okay	Write	8-beat incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst

2) Project Management Needs

a) Required tools.

Our verification plan will require a set of tools to support the simulation, testbench creation, and analysis of results. Here's a breakdown:

a. *Simulation Tool - QuestaSim:*
   QuestaSim is a comprehensive tool that supports advanced verification methodologies and provides extensive debugging features. It supports both VHDL and Verilog languages, which makes it suitable for our project as we are using SystemVerilog for our design. QuestaSim will be used to run the simulations of our testbenches against the RTL design.

b. *Testbench Language - SystemVerilog (SV):*
   SystemVerilog is a hardware description and hardware verification language used to model, design, and verify electronic systems. It extends the Verilog hardware description language with powerful verification features. We will use SystemVerilog to write our testbenches due to its advanced verification capabilities, such as classes, randomization, and functional coverage, which will help us create robust and comprehensive tests for our design.

c. *Assertion Language - SystemVerilog Assertions (SVA):*
   SystemVerilog Assertions (SVA) is a part of the SystemVerilog language used for specifying assertions about the behavior of a design. Assertions are used to specify the expected behavior of a design, and they provide a powerful way to detect and locate errors. We will use SVA to specify assertions in our testbenches, which will help us ensure that our design behaves as expected.

d. *Version Control System:*
   A version control system is essential for managing the different versions of our design and testbench files. It allows us to track changes, revert to previous versions if needed, and collaborate effectively as a team. We can use a tool like Git for this purpose.

e. *Coverage Analysis Tool:*
   A coverage analysis tool will be needed to measure the effectiveness of our tests. This tool should be able to analyze both code coverage and functional coverage. QuestaSim provides built-in coverage analysis capabilities that we can use.

f. *Documentation Tool:*
   A documentation tool will be needed to maintain the documentation of our verification plan, test cases, and results. We plan to use both Microsoft Word and Google Docs for this purpose.

By using these tools, we can effectively manage our project, create comprehensive testbenches, run simulations, analyze the results, and ensure that our AHB-APB bridge design meets all its specifications and works as expected.

b) Risks and Dependencies.

- **Risks:** The main risks in our project include potential bugs in the design, delays in the verification process due to unforeseen complexities or difficulties, and the learning curve associated with using advanced verification methodologies and tools. We also have a very tight schedule. To mitigate these risks, we will use a systematic and thorough verification approach, start our work early, and seek help when needed.

- **Dependencies:** Our project depends on the availability of the QuestaSim simulator, the completion of the RTL design, and the availability of the ARM documentation for the AHB-APB bridge. We will need to coordinate with the design team to ensure that the design is completed on schedule and that any changes to the design are communicated to us promptly.

c) Resources.

- **People:** Our team consists of **two graduate students.** We will need to divide the work between us and coordinate our efforts to ensure that all tasks are completed on time. We may also need to seek help from our professors or colleagues if we encounter difficulties.

- **Computing:** We will need sufficient computing resources to run our simulations. This includes a computer with enough processing power and memory to run QuestaSim and handle our design and testbenches.

- **Licenses:** We will need licenses for QuestaSim and any other commercial tools that we use. We will need to ensure that these licenses are available and valid for the duration of our project.

d) Schedule.

- **Environment creation, debugging, regression:** We will need to allocate time for creating our verification environment, debugging our testbenches, and running regression tests. We will start with the creation of the SV testbench, which is due on 7/31/2023, and then move on to the UVM testbench, which is due on 8/17/2023.

  First week (7/3/2023):
  Choose the RTL design, and write a basic testbench to verify it.
  Second Week (7/10/2023):
  Analyze and understand the RTL design, and start reading the ARM AMBA specification document.
  Third Week (7/17/2023):
  Layout the verification plan (high level and low level) and start preparing the testcases.
  Fourth Week (7/24/2023):
  Work on constructing the SV testbench environment.
  Fifth Week (7/31/2023):

Complete the SV verification environment components and deliver it. (submission).

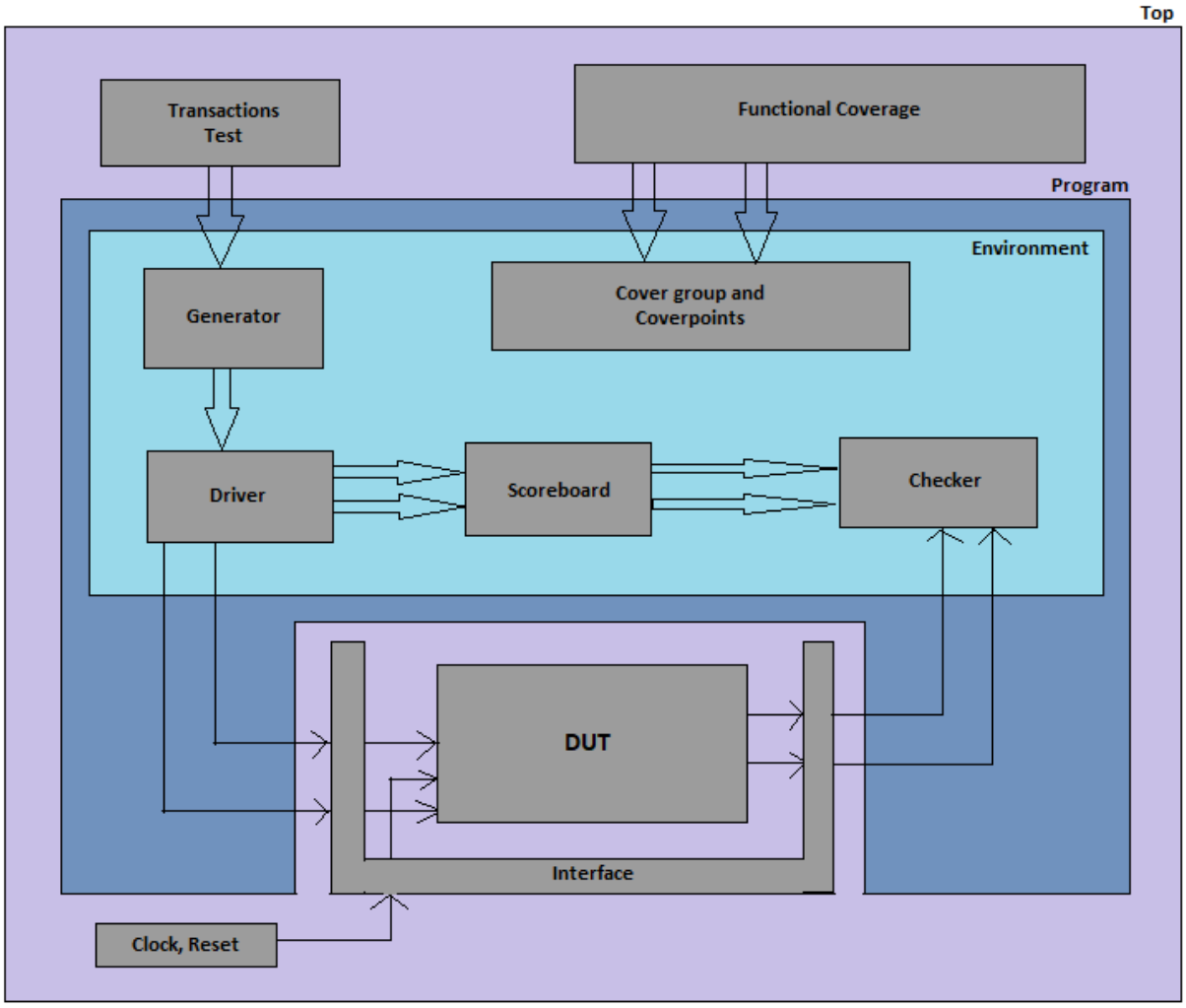
Six Week (8/7/2023):

Learn and practice the UVM environment and start transitioning the verification environment from the basic SV environment to UVM-based environment.

Seventh Week (8/17/2023):

Complete the UVM verification environment components and deliver it. (submission).

- **Monitoring:** We will need to monitor our progress regularly to ensure that we are on track. We will look at indicators such as the number of tests passed, the coverage achieved, and the number of bugs found and fixed. We will also set expectations for our progress, such as expecting the number of problems found per week to drop off as we get further into the project. We can use our history of progress to predict our future progress and adjust our plans as needed.



Summary

Our verification plan is based on grey-box testing methodologies and transaction-based reference model checking. The grey-box testing will allow us to examine the internal workings of the system and gain better controllability and observability, especially during debugging.

The plan includes a detailed test case matrix, which outlines a range of scenarios, including different types of transactions, data transfer sizes, and burst types. The matrix also includes tests for corner cases and error conditions. Each test case is designed to verify a specific function or feature of the bridge, and the expected outputs for each test case are based on the specifications of the AHB-APB bridge.

In terms of resources, we have allocated two graduate students, computing resources, and software licenses for the project. We have also identified potential risks and dependencies, including potential bugs in the design and delays in the verification process.

In conclusion, our verification plan provides a systematic and thorough approach to verifying the functionality and performance of the AHB-APB bridge. By following this plan, we aim to ensure that the bridge operates correctly under all possible conditions and scenarios, thereby contributing to the overall reliability and performance of our system.

Appendix: Testcase matrix

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10
Test Case ID	HTRANS	HSIZE	HBURST	HRESP	Operation	Description	Inputs	Expected Outputs	Function Being Tested
1	NONSEQUENTIAL	Byte	Single Transfer	Okay	Read	Single byte read operation	HADDR, HWDATA	PRDATA, HREADY	Read Operation
2	NONSEQUENTIAL	Halfword	Single Transfer	Okay	Write	Single halfword write operation	HADDR, HWDATA	HREADY	Write Operation
3	NONSEQUENTIAL	Word	Single Transfer	Okay	Read	Single word read operation	HADDR, HWDATA	PRDATA, HREADY	Read Operation
4	NONSEQUENTIAL	Byte	Single Transfer	Error	Write	Single byte write operation with error	HADDR, HWDATA	HREADY, HRESP=Error	Write Operation with Error
5	NONSEQUENTIAL	Halfword	Incrementing Burst	Okay	Read	Incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
6	NONSEQUENTIAL	Word	Incrementing Burst	Okay	Write	Incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
7	NONSEQUENTIAL	Byte	4-beat Wrapping Burst	Okay	Read	4-beat wrapping burst read operation with byte size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
8	NONSEQUENTIAL	Halfword	4-beat Wrapping Burst	Okay	Write	4-beat wrapping burst write operation with halfword size	HADDR, HWDATA	HREADY	Write Operation with Burst
9	NONSEQUENTIAL	Word	4-beat Wrapping Burst	Okay	Read	4-beat wrapping burst read operation with word size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
10	NONSEQUENTIAL	Byte	4-beat Incrementing Burst	Okay	Write	4-beat incrementing burst write operation with byte size	HADDR, HWDATA	HREADY	Write Operation with Burst
11	NONSEQUENTIAL	Halfword	4-beat Incrementing Burst	Okay	Read	4-beat incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
12	NONSEQUENTIAL	Word	4-beat Incrementing Burst	Okay	Write	4-beat incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
13	NONSEQUENTIAL	Byte	8-beat Wrapping Burst	Okay	Read	8-beat wrapping burst read operation with byte size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
14	NONSEQUENTIAL	Halfword	8-beat Wrapping Burst	Okay	Write	8-beat wrapping burst write operation with halfword size	HADDR, HWDATA	HREADY	Write Operation with Burst
15	NONSEQUENTIAL	Word	8-beat Wrapping Burst	Okay	Read	8-beat wrapping burst read operation with word size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
16	NONSEQUENTIAL	Byte	8-beat Incrementing Burst	Okay	Write	8-beat incrementing burst write operation with byte size	HADDR, HWDATA	HREADY	Write Operation with Burst
17	NONSEQUENTIAL	Halfword	8-beat Incrementing Burst	Okay	Read	8-beat incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
18	NONSEQUENTIAL	Word	8-beat Incrementing Burst	Okay	Write	8-beat incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
19	SEQUENTIAL	Byte	Single Transfer	Okay	Read	Sequential single byte read operation	HADDR, HWDATA	PRDATA, HREADY	Read Operation
20	SEQUENTIAL	Halfword	Single Transfer	Okay	Write	Sequential single halfword write operation	HADDR, HWDATA	HREADY	Write Operation
21	SEQUENTIAL	Word	Single Transfer	Okay	Read	Sequential single word read operation	HADDR, HWDATA	PRDATA, HREADY	Read Operation
22	SEQUENTIAL	Byte	Single Transfer	Error	Write	Sequential single byte write operation with error	HADDR, HWDATA	HREADY, HRESP=Error	Write Operation with Error
23	SEQUENTIAL	Halfword	Incrementing Burst	Okay	Read	Sequential incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
24	SEQUENTIAL	Word	Incrementing Burst	Okay	Write	Sequential incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
25	SEQUENTIAL	Byte	4-beat Wrapping Burst	Okay	Read	Sequential 4-beat wrapping burst read operation with byte size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
26	SEQUENTIAL	Halfword	4-beat Wrapping Burst	Okay	Write	Sequential 4-beat wrapping burst write operation with halfword size	HADDR, HWDATA	HREADY	Write Operation with Burst
27	SEQUENTIAL	Word	4-beat Wrapping Burst	Okay	Read	Sequential 4-beat wrapping burst read operation with word size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
28	SEQUENTIAL	Byte	4-beat Incrementing Burst	Okay	Write	Sequential 4-beat incrementing burst write operation with byte size	HADDR, HWDATA	HREADY	Write Operation with Burst
29	SEQUENTIAL	Halfword	4-beat Incrementing Burst	Okay	Read	Sequential 4-beat incrementing burst read operation with halfword size	HADDR, HWDATA	PRDATA, HREADY	Read Operation with Burst
30	SEQUENTIAL	Word	4-beat Incrementing Burst	Okay	Write	Sequential 4-beat incrementing burst write operation with word size	HADDR, HWDATA	HREADY	Write Operation with Burst
31	NONSEQUENTIAL	Byte	Single Transfer	Okay	Reset	Reset during a single byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
32	NONSEQUENTIAL	Halfword	Single Transfer	Okay	Reset	Reset during a single halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
33	NONSEQUENTIAL	Word	Single Transfer	Okay	Reset	Reset during a single word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
34	NONSEQUENTIAL	Byte	Incrementing Burst	Okay	Reset	Reset during an incrementing burst byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
35	NONSEQUENTIAL	Halfword	Incrementing Burst	Okay	Reset	Reset during an incrementing burst halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
36	NONSEQUENTIAL	Word	Incrementing Burst	Okay	Reset	Reset during an incrementing burst word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
37	NONSEQUENTIAL	Byte	4-beat Wrapping Burst	Okay	Reset	Reset during a 4-beat wrapping burst byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
38	NONSEQUENTIAL	Halfword	4-beat Wrapping Burst	Okay	Reset	Reset during a 4-beat wrapping burst halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
39	NONSEQUENTIAL	Word	4-beat Wrapping Burst	Okay	Reset	Reset during a 4-beat wrapping burst word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
40	NONSEQUENTIAL	Byte	4-beat Incrementing Burst	Okay	Reset	Reset during a 4-beat incrementing burst byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
41	NONSEQUENTIAL	Halfword	4-beat Incrementing Burst	Okay	Reset	Reset during a 4-beat incrementing burst halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
42	NONSEQUENTIAL	Word	4-beat Incrementing Burst	Okay	Reset	Reset during a 4-beat incrementing burst word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
43	NONSEQUENTIAL	Byte	8-beat Wrapping Burst	Okay	Reset	Reset during an 8-beat wrapping burst byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
44	NONSEQUENTIAL	Halfword	8-beat Wrapping Burst	Okay	Reset	Reset during an 8-beat wrapping burst halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
45	NONSEQUENTIAL	Word	8-beat Wrapping Burst	Okay	Reset	Reset during an 8-beat wrapping burst word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
46	NONSEQUENTIAL	Byte	8-beat Incrementing Burst	Okay	Reset	Reset during an 8-beat incrementing burst byte transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
47	NONSEQUENTIAL	Halfword	8-beat Incrementing Burst	Okay	Reset	Reset during an 8-beat incrementing burst halfword transfer	HADDR, HWDATA, HRESET	HREADY, HRESET=1	Reset during Transfer
48	NONSEQUENTIAL	Word	8-beat Incrementing Burst	Okay	Reset	Reset during an 8-beat incrementing burst word transfer	HADDR, HWDATA, HRESET	PRDATA, HREADY, HRESET=1	Reset during Transfer
49	SEQUENTIAL	Byte	Single Transfer	Error	Error Handling	Error during a single byte transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
50	SEQUENTIAL	Halfword	Single Transfer	Error	Error Handling	Error during a single halfword transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
51	SEQUENTIAL	Word	Single Transfer	Error	Error Handling	Error during a single word transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
52	SEQUENTIAL	Byte	Incrementing Burst	Error	Error Handling	Error during an incrementing burst byte transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
53	SEQUENTIAL	Halfword	Incrementing Burst	Error	Error Handling	Error during an incrementing burst halfword transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
54	SEQUENTIAL	Word	Incrementing Burst	Error	Error Handling	Error during an incrementing burst word transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
55	SEQUENTIAL	Byte	4-beat Wrapping Burst	Error	Error Handling	Error during a 4-beat wrapping burst byte transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
56	SEQUENTIAL	Halfword	4-beat Wrapping Burst	Error	Error Handling	Error during a 4-beat wrapping burst halfword transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
57	SEQUENTIAL	Word	4-beat Wrapping Burst	Error	Error Handling	Error during a 4-beat wrapping burst word transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
58	SEQUENTIAL	Byte	4-beat Incrementing Burst	Error	Error Handling	Error during a 4-beat incrementing burst byte transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
59	SEQUENTIAL	Halfword	4-beat Incrementing Burst	Error	Error Handling	Error during a 4-beat incrementing burst halfword transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling
60	SEQUENTIAL	Word	4-beat Incrementing Burst	Error	Error Handling	Error during a 4-beat incrementing burst word transfer	HADDR, HWDATA	HREADY, HRESP=Error	Error Handling