

APB WITH 2 SLAVE

Table of Contents

[1. INTRODUCTION 3](#_Toc25452)

1.1 Key Features 3

1.2 Block Diagram 3

1.3 APB Pin Description 4

1.4 Operation of APB 5

[2. ARCHITECTURE 7](#_Toc26066)

[2.1 Standard Architecture 7](#_Toc7373)

[2.2 Testbench Architecture 7](#_Toc10629)

[2.3 DUT 8](#_Toc27093)

[3. TESTBENCH COMPONENTS 9](#_Toc17199)

[1. apb\_interface 9](#_Toc16182)

[2. apb\_top 9](#_Toc15971)

[3. apb\_test](#_Toc54) 9

[4. apb\_environment](#_Toc13803) 10

[5. apb\_agent 1](#_Toc26449)0

[6. apb\_sequence\_item 1](#_Toc29044)1

7. apb\_sequence 11

8. apb\_sequencer 11

9. apb\_driver 11

10. apb\_monitor 12

11. apb\_scoreboard 12

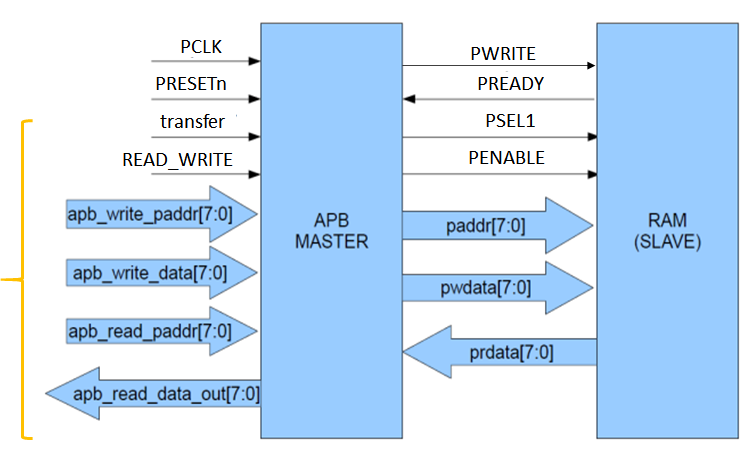
12. apb\_coverage 13

[4. TESTCASES 1](#_Toc12011)4

1. **INTRODUCTION**

The Advanced Peripheral Bus (APB) is part of the Advanced Micro-controller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipe lined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.APB is a non pipe lined structure.

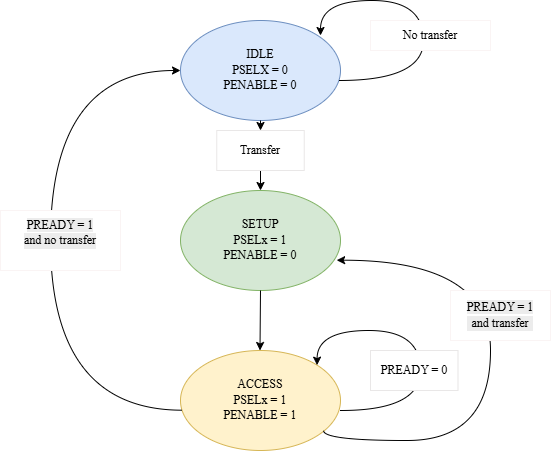
1. Key Features
2. It supports addresses up to 32 bit wide.
3. APB4 support for write strobe signal to enable sparse data transfer on the write data bus.
4. Single Master - Multiple Slaves.
5. Programmable Wait state insertion.
6. Slave supports fine grain control of response per address or per transfer.
7. Simple Interface.
8. Suitable for many Peripherals.
9. In APB every transfer takes at least two cycles(Setup Phase and Access Phase).
10. Block Diagram

[](https://user-images.githubusercontent.com/82434808/122651062-0bc74980-d154-11eb-9737-591e928a734e.png)

1. APB Pin Description

|  |  |  |  |
| --- | --- | --- | --- |
| **SIGNAL** | **SOURCE** | **DESCRIPTION** | **WIDTH(BIT)** |
| Transfer | System Bus | APB enable signal. If high APB is activated else APB is disabled | 1 |
| PCLK | Clock Source | All APB functionality occurs at a rising edge. | 1 |
| PRESETn | System Bus | An active low signal. | 1 |
| PADDR | APB bridge | The APB address bus can be up to 32 bits. | 8 |
| PSEL1 | APB bridge | There is a PSEL for each slave. It’s an active high signal. | 1 |
| PENABLE | APB bridge | It indicates the 2nd cycle of a data transfer. It’s an active high signal. | 1 |
| PWRITE | APB bridge | Indicates the data transfer direction. PWRITE=1 indicates APB write access(Master to slave) PWRITE=0 indicates APB read access(Slave to master) | 1 |
| PREADY | Slave Interface | This is an input from Slave. It is used to enter the access state. | 1 |
| PSLVERR | Slave Interface | This indicates a transfer failure by the slave. | 1 |
| PRDATA | Slave Interface | Read Data. The selected slave drives this bus during reading operation | 8 |
| PWDATA | Slave Interface | Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is high. | 8 |

1. Operation of APB



1. Idle State:

* PSELx = 0, PENABLE = 0
* This is the default or inactive state. No transfer is initiated here
* The system will remain in the IDLE state if PREADY = 1 and no transfer is required
* Transition occurs to the SETUP state if a transfer request is initiated (trigger not explicitly shown but implied by the transition arrow)

1. Setup State:

* PSELx = 1, PENABLE = 0
* In this state, the peripheral select signal (PSELx) is asserted, indicating the target peripheral is selected for communication.
* The system is preparing for the actual data transfer
* Transition to the ACCESS state occurs when PREADY = 0, meaning the bus is not yet ready to complete the transfer

1. Access State:

* PSELx = 1, PENABLE = 1
* This state represents the active transfer phase, where both PSELx and PENABLE are asserted.
* The data transfer occurs during this state
* If PREADY = 1, indicating the bus is ready, and the transfer completes, the system transitions back to the IDLE state
* If PREADY = 0, the system stays in the ACCESS state, waiting for the bus to be ready

1. **ARCHITECTURE**
2. Standard Architecture

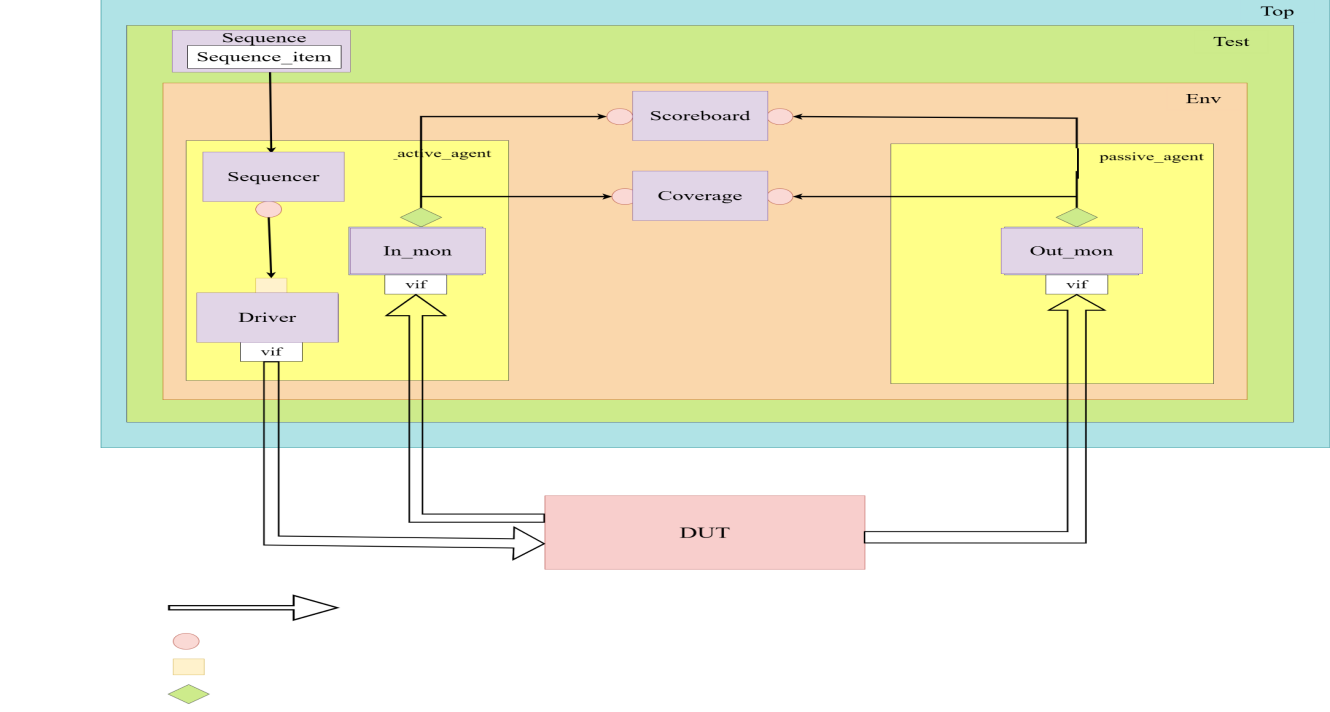


Fig 2.1.1 Standard Architecture

1. Testbench Architecture

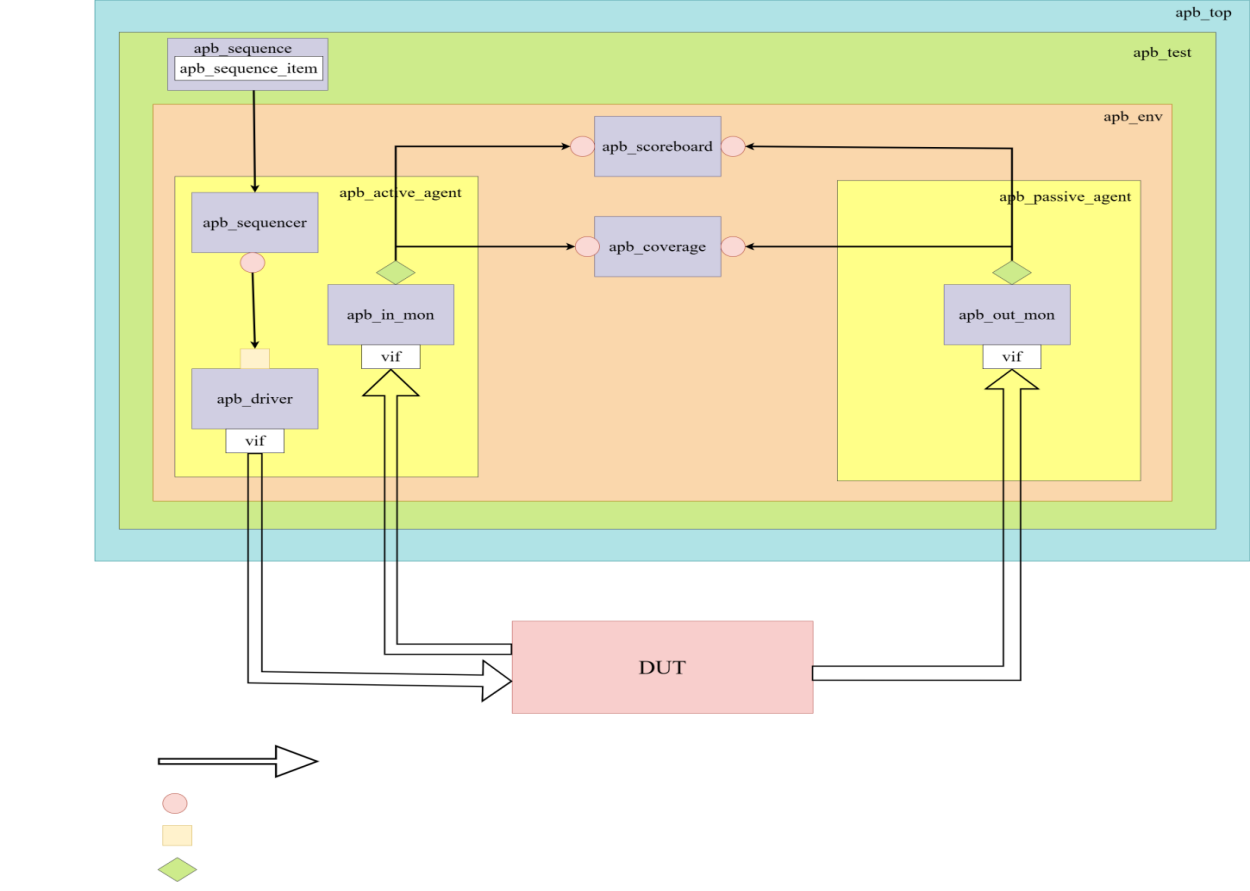
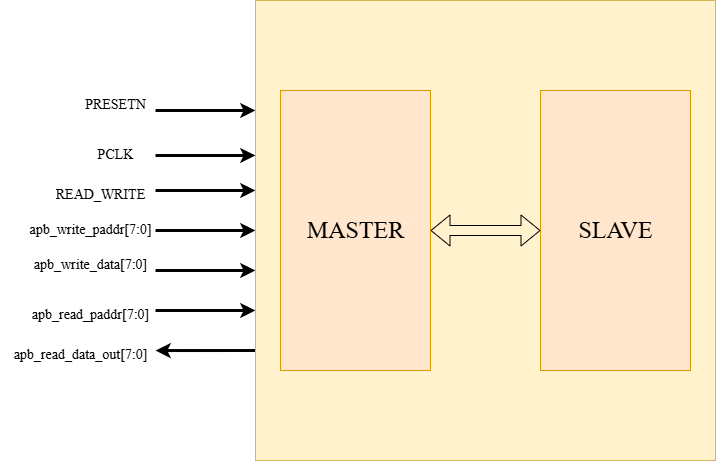
****

Fig 2.2.1 Test bench Architecture

1. DUT



1. **TESTBENCH COMPONENTS**
2. apb\_interface

* Interface module declaration
* Signal Declaration
* Clocking Block for apb\_driver, apb\_monitor
* Modport Declaration for apb\_driver, apb\_monitor

1. apb\_top

* Include UVM packages
* Include UVM macros
* Import UVM package
* Instantiate the DUT
* Create Virtual Interface
* Instantiate vif
* Pass the vif to the UVM environment using `uvm\_config\_db.
* Clock and reset generation
* Start the UVM test run\_test()
* Waveform generation

1. apb\_test

* Define apb\_test class
* Extend apb\_test from uvm\_test
* Use Macros to register the apb\_test with UVM factory
* Create class constructor
* Declare a handle for apb\_environment
* Implement the Build Phase
* Instantiate the apb\_environment in the build phase
* Check for the end of elaboration
* Implement the report Phase
* Generate report
* Print summary

1. apb\_environment

* Define apb\_environment class
* Extend apb\_environment from UVM\_environment
* Use Macros to register the apb\_env with UVM factory
* Create a class constructor
* Declare and Instantiate agents
* Declare a handle for active agent and passive agent
* Declare and Instantiate Scoreboard
* Declare and Instantiate Coverage
* Implement the build phase
* Create instance of all the components
* Implement the connect phase
* Connect agents and scoreboard
* Setup analysis port the transfer data from monitor to scoreboard and coverage

1. apb\_agent

* Define apb\_agent class
* Extend apb\_agent from uvm\_agent
* Use Macros to register the apb\_agent with UVM factory
* Create a class constructor
* Declare an handle for apb\_sequencer, apb\_monitor and apb\_driver
* Declare a flag “is\_active” to determine whether the agent is active or passive
* Implement the build phase
* Instantiate the monitor and sequencer
* The build\_phase of the agent class checks this flag and decides whether to instantiate the driver
* Implement the connect phase
* Connect sequencer to driver for active agent
* Connect monitor’s analysis port
* We are making use of active and passive agent

1. apb\_sequence\_item

* User defined sequence items obtained from uvm\_sequence\_item class
* Input variables are declared with the rand keyword
* Output variables are not declared as randomizable
* The class includes constraints to guide randomization of inputs

1. apb\_sequence

* The operation which is intended to be done by sequence is defined inside a body method
* This includes 6 methods
* Create\_item
* Wait\_for\_grant()
* randomize()
* send\_request()
* wait\_for\_item\_done()
* get\_response()
* For each test cases same steps are repeated

1. apb\_sequencer

* A user-defined sequencer is extended uvm\_sequencer
* Use of uvm macros for factory registration
* Usage of class constructor

1. apb\_driver

* class name: apb\_driver
* virtual interface handle
* function build\_phase()
* `uvm\_config\_db is used to retrieve a configuration setting from the UVM configuration database
* A task run\_phase()
* Methods to retrieve the next sequence item from the sequencer
* “seq\_item\_port” to connect driver to the sequencer
* “get\_next\_item()” to fetch the next sequence item from the sequencer queue
* “Drive()” task is called
* “Item\_done()” method indicates to the sequencer that the current sequence item processing has been completed
* virtual task driver()
* “drive()” task is used to drive the values of the sequence item onto the DUT through the interface

1. apb\_monitor

* Create a user-defined monitor class extended from uvm\_monitor and register it in the uvm factory
* Declare analysis port to broadcast the sequence items or transactions
* Declare virtual interface handle to retrieve actual interface handle
* Write standard class constructor create an instance for sequence\_item
* Implement build\_phase and get the interface handle from the configuration database
* Implement run\_phase to sample DUT interface using a virtual interface handle and translate into transactions
* The write() method sends transactions to the collector component
* In apb\_in\_monitor input signals are captured
* In apb\_out\_monitor DUT signals are captured

1. apb\_scoreboard

* class name: apb\_scoreboard
* Establish a fifo to hold the value of seq\_item
* analysis port “item\_collected\_export” is the port through which the sequence items are sent to the apb\_scoreboard analysis component
* function build\_phase ()
* creates an instance of the “uvm\_analysis\_imp” class
* virtual function write ()
* Function used to handle the incoming “sequence\_item” objects i.e transactions
* virtual task run\_phase ()

1. apb\_coverage

* define apb\_coverage class extended from uvm\_subscriber
* Define a coverage group that contains coverpoints and cross coverage
* Coverage Group Constructor Is created to initialize the coverage group
* write() method to collect data
* Register the apb\_coverage class with the UVM factory

1. **TESTCASES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl No.** | **Features** | **Sub Feature** | **Description** | **Status** | **Test Case Name** |
| 1 | Write operation | READ\_WRITE = 1 signal | Write data to the specified address | Pass/Fail | apb\_write |
| 2 | Read operation | READ\_WRITE = 0 signal | Read data from the specified address | Pass/Fail | apb\_read |
| 3 | Reset | Active low reset | When reset is active low all signals are set to default values | Pass/Fail | apb\_reset |
| 4 | Write & Read op | Simultaneous write & read | Perform write to an address followed by a read from it | Pass/Fail | apb\_write\_read |
| 5 | Transfer control | Transfer signal = 0 | If the transfer signal goes low in the middle of transaction then apb is deactivated | Pass/Fail | apb\_transfer\_low |
| 6 | Slave selection | Based on 9th bit of address | If apb\_write\_paddr[8] = 0, then Slave1 is selected If apb\_write\_paddr[8] = 1, then Slave2 is selected | Pass/Fail | apb\_slave\_select |
| 7 | Read from different address |  | The data is read from different address but is written in some other address | Pass/Fail | apb\_read\_diff\_addr |