2021

APB AVIP Project

USER GUIDE

Contents

List of Tables

List of Figures

Chapter 1

Introduction

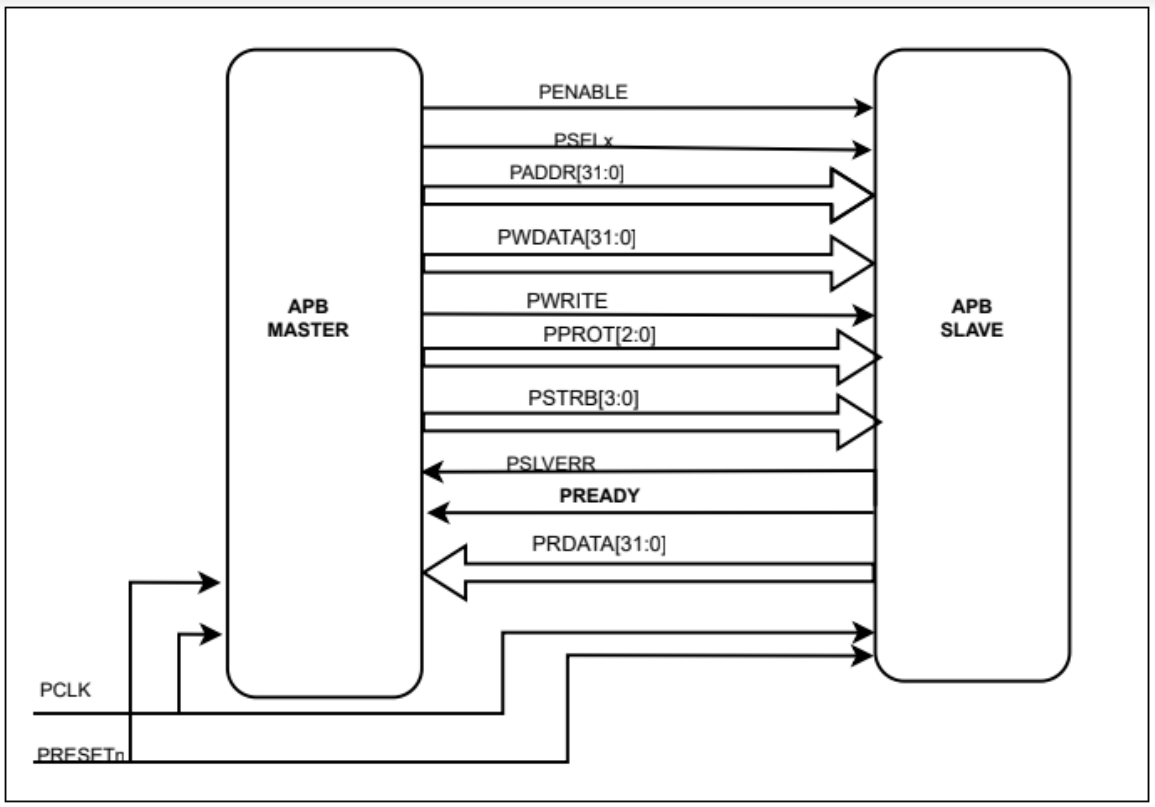


Fig: 1.1 APB AVIP

Master avip can communicate with the slave avip via APB interface. Master avip and slave avip works on both simulator and emulator. To know more about avip, please go through the link : *SystemVerilog Testbench Acceleration | Acceleration*

1.1 APB master avip

APB master avip starts the test in the hvl top and starts the randomised sequences in base_test and will pass the paddr, pwrite, pselx and pwdata to APB master_driver proxy using uvm sequencer and driver handshake. The APB master driver bfm gets the paddr, pwrite, pselx and pwdata using inbound communication. The APB master driver bfm sends the paddr, pwrite, pselx and pwdata that is sampled in APB master driver bfm and sends it back to the APB master driver proxy. The APB master monitor bfm samples the master_tx and slave_tx received from the APB interface and sends it to the APB master monitor proxy. The sampled data received by APB master monitor bfm is sent to the APB master scoreboard and APB

master coverage. APB master scoreboard compares the driven data and sampled data. APB master coverage is used to check the functional coverage of APB master.

To know more about inbound and outbound communication, please go through this link : *Inbound and Outbound Communication*

1.2 APB slave avip

APB slave avip starts the test in the hvl top and starts the randomised sequences in base_test and will pass the prdata, pslverr and pready to APB slave_driver proxy using uvm sequencer and driver handshake. The APB slave driver bfm gets prdata,pslverr and pready using inbound communication. The APB slave driver bfm sends the prdata, pslverr and pready that is sampled in APB slave driver bfm and sends it back to the APB slave driver proxy. The APB slave monitor bfm samples samples the master_tx and slave_tx received from the APB interface and sends it to the APB slave monitor proxy. The sampled data received by APB slave monitor bfm is sent to the APB slave scoreboard and APB slave coverage. APB slave scoreboard compares the driven data and sampled data. APB slave coverage is used to check the functional coverage of APB slave.

To know more about inbound and outbound communication, please go through this link : *Inbound and Outbound Communication*

1.3 APB Interface

APB interface has the following interface pin level signals :

Signals	Source	Description
pclk	Clock source	Clock. The rising edge of PCLK times all transferon the APB.
preset_n	System bus equivalent	Reset. The APB reset signal is active low. This signal is normally connected directly to the system bus reset signal
paddr	APB bridge	Address. This is the APB address bus.It can be up to 32 bits wide and is a data access or an instruction access.
pprot	APB bridge	Protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.

pselx	APB bridge	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a pselx signal for each slave.
penable	APB bridge	Enable. This signal indicates the second and subsequent cycle of an APB transfer.
pwrite	APB bridge	Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW.
pwdata	APB bridge	Write data. This bus is driven by the peripheral bus bridge unit during the write cycle when pwrite is HIGH. This bus can be up to 32 bits wide.
pstrb	APB bridge	Write strobes. This signal indicates when byte lanes to update during a write transfer. There is one write strobe for each eight bits of the write data bus. Therefore, pstrb[n] corresponds to pwdata[(8n+7):(8n)]. Write strobes must not be active during a read transfer.
pready	Slave interface	Ready. The Slave uses this signal to extend an APB transfer.
prdata	Slave interface	Read Data.The selected slave drives this bus during read cycles when pwrite is LOW. This bus can be up to 32-bits wide.
pslverr	Slave interface	This signal indicates a transfer failure. APB peripherals are not required to support the pslverr pin. This is true for both existing and new APB peripheral designs. Where a peripheral does not include this PIN then the appropriate input to the APB bridge is tied LOW.

To know more about the APB interface signals, please go to section *3.1 Pin Interface*.

1.4 apb_compile.f file

This file contains the following things :

1. All the directories needed
2. All the packages we needed

3. All the modules written
4. All the bfm interfaces written
5. The APB interface

- If you want to add any file in the project, please add the file or folder in the apb_compile.f file to make it compiled.
- If you want to add a class based component or object, you have to add that file in the respective package file and then make sure to mention the directory and path in the apb_compile.f file.
- If you want to add a module/interface or any static component, then mention the file name along with the path of the file.
- How to add :
    1. To include directory: +incdir<path_of_the_folder>
    2. To include file use file_path/file_name.extension
    3. / is used to force a new line
- Current apb_compile.f file consists of all files directories and Packages mentioned in fig. 1.2, fig. 1.3 and fig. 1.4.

a. Directories included :

```
+incdir+../../src/globals/
+incdir+../../src/hvl_top/test/sequences/master_sequences/
+incdir+../../src/hvl_top/master/
+incdir+../../src/hdl_top/master_agent_bfm/
+incdir+../../src/hvl_top/env/virtual_sequencer/
+incdir+../../src/hvl_top/test/virtual_sequences/
+incdir+../../src/hvl_top/env
+incdir+../../src/hvl_top/slave
+incdir+../../src/hvl_top/test/sequences/slave_sequences/
+incdir+../../src/hvl_top/test
+incdir+../../src/hdl_top/slave_agent_bfm
+incdir+../../src/hdl_top/apb_interface
```

Fig: 1.2 Directories included in apb_compile.f file

b. Packages included:

```
../../src/globals/apb_global_pkg.sv
../../src/hvl_top/master/apb_master_pkg.sv
../../src/hvl_top/slave/apb_slave_pkg.sv
../../src/hvl_top/test/sequences/master_sequences/apb_master_seq_pkg.sv
../../src/hvl_top/test/sequences/slave_sequences/apb_slave_seq_pkg.sv
../../src/hvl_top/env/apb_env_pkg.sv
../../src/hvl_top/test/virtual_sequences/apb_virtual_seq_pkg.sv
../../src/hvl_top/test/apb_base_test_pkg.sv
```

Fig: 1.3 Packages included in apb_compile.f file

c. Static files included:

```
../../src/hdl_top/apb_if/apb_if.sv
../../src/hdl_top/master_agent_bfm/apb_master_driver_bfm.sv
../../src/hdl_top/master_agent_bfm/apb_master_monitor_bfm.sv
../../src/hdl_top/master_agent_bfm/apb_master_agent_bfm.sv
../../src/hdl_top/slave_agent_bfm/apb_slave_driver_bfm.sv
../../src/hdl_top/slave_agent_bfm/apb_slave_monitor_bfm.sv
../../src/hdl_top/slave_agent_bfm/apb_slave_agent_bfm.sv
../../src/hdl_top/hdl_top.sv
```

Fig: 1.4 static files included in apb_compile.f file

In Makefile, we include the apb_compile.f file to compile all the files included.
Command used : *irun -f apb_compile.f +UVM_TEST_NAME=<test_name> +uvm_verbosity=UVM_HIGH.*

```
compile:
  make clean_compile;
  make clean_simulate;
  vlib work;
  vlog -sv \
  +acc \
  +cover \
  +fcover \
  -l apb_compile.log \
  -f ../apb_compile.f
```

Fig: 1.5 APB_compile.f used in makefile for questa_sim tool

```
irun -c \
  -clean \
  -elaborate \
  -coverage a \
  -access +rwc \
  -64 \
  -sv \
  -uvm \
  +access+rw \
  -f ../apb_compile.f \
  -l apb_compile.log \
  -top worklib.hdl_top:sv \
  -top worklib.hvl_top:sv \
  -nclibdirname INCA_libs \
  -SVA
```

Fig 1.6 APB_compile.f used in makefile for cadence tool

Chapter 2

Architecture

APB AVIP Testbench Architecture has divided into the two top modules as HVL and HDL top as shown in below fig 2.1

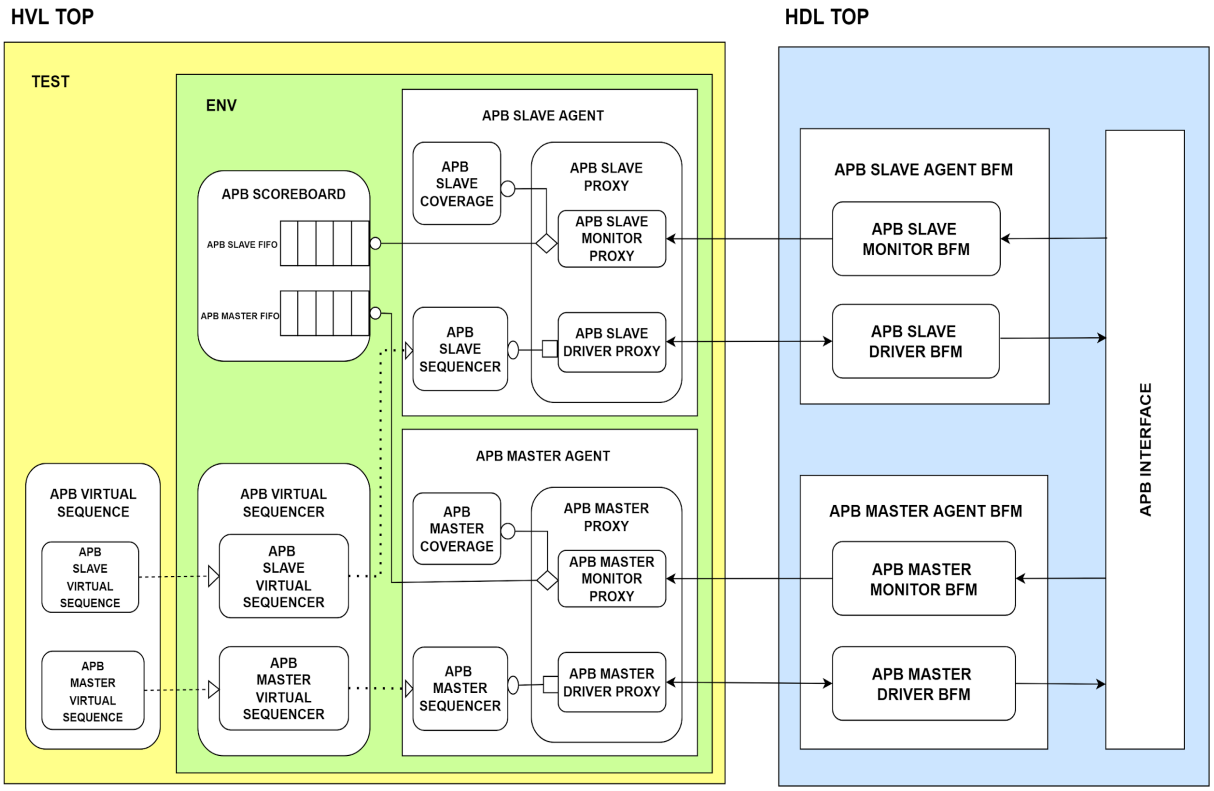


Fig 2.1 apb avip Architecture

The whole idea of using Accelerated VIP is to push the synthesizable part of the testbench into the separate top module along with the interface and it is named as HDL TOP and the unsynthesizable part is pushed into the HVL TOP it provides the ability to run the longer tests quickly. This particular testbench can be used for the simulation as well as the emulation based on mode of operation.

HVL TOP has the design which is untimed and the transactions flow from both master virtual sequence and slave virtual sequence onto the APB I/F through the BFM Proxy and BFM and gets the data from monitor BFM and uses the data to do checks using scoreboard and coverage.

HDL TOP consists of the design part which is timed and synthesizable, Clock and reset signals are generated in the HDL TOP. Bus Functional Models (BFMs) i.e synthesizable part of drivers and monitors are present in HDL TOP, BFMs also have the back pointers to it's proxy to call non - blocking methods which are defined in the proxy.

Tasks and functions within the drivers and monitors which are called by the driver and monitor proxy inside the HVL. This is how the data is transferred between the HVL TOP and HDL TOP.

HDL and HVL uses the transaction based communication to enable the information rich transactions and since clock is generated within the HDL TOP inside the emulator it allows the emulator to run at full speed.

Chapter 3

Steps to run Test Cases

3.1 Git steps

1. Checking for git, open the terminal type the command

   git version

   The output will either tell you which version of Git is installed or alert you that git is an unknown command. If it's an unknown command, install Git using following link *guide to install git in other platforms*
2. Copy the ssh public key and do the clone of the APB_avip repository in the terminal

   find the apb_avip GitHub repository here

   git clone git@github.com:mirafra-software-technologies/apb_avip.git
3. After cloning, change the directory to the cloned repository

   cd apb_avip
4. After cloning you will be in the main branch i.e, the production branch

   git branch
5. Do the pull for the cloned repository to be in sync

   git pull origin main
6. Fetch all branches in the apb_avip repository

   git fetch
7. Check all branches present in the apb_avip repository

   git branch -a
8. To switch from the main branch to another branch

   git checkout origin <branch_name>
9. Do the pull for the cloned repository to be in sync

   git pull origin <branch_name>

Note: To run any test case you should be inside the cloned directory i.e, apb_avip [apb_avip is considered as root path]

3.2 Mentor's Questasim

1. Change the directory to questasim directory where the makefile is present

   Path for the mentioned directory is apb_avip/sim/questasim

   Note: To Compile, simulate, regression and for coverage you must be in the specified path i.e, *apb_avip/sim/questasim*

2. To view the usage for running test cases, type the command

   make

   Fig 3.1 shows the usage to compile, simulate, and regression

```
------------------------------------------------------------------
------------------------- Usage ----------------------------------

make target <options> <variable>=<value>

To compile use:
make compile

To simulate individual test:
make simulate test=<test_name> uvm_verbosity=<VERBOSITY_LEVEL>

Example::
make simulate test=base_test uvm_verbosity=UVM_HIGH

To run regression:
make regression testlist_name=<regression_testlist_name.list>
```

Fig 3.1 Usage of the make command

3.2.1 Compilation

1. Use the following command to compile

   make compile

2. Open the log file *apb_compile.log* to view the compiled files
   gvim apb_compile.log

3.2.2 Simulation

1. After compilation, use the following command to simulate individual test cases

   make simulate test=<test_name> uvm_verbosity=<VERBOSITY_LEVEL>

   Example:

2. To view the log file
   gvim <test_name>/<test_name>.log

   Ex: *gvim apb_8b_write_test/apb_8b_write_test.log*

Note: The path for the log file will be displayed in the simulation report along with the name of the simulated test

3. To view waveform

   vsim -view <test_name>/waveform.wlf &

   Ex: *vsim -view apb_8b_write_test/waveform.wlf &*

   Note: The command to view the waveform will be displayed in the simulation report along with the name of the simulated test

4. As you run the above command, the new WLF Questasim window will appear as shown in fig 3.2

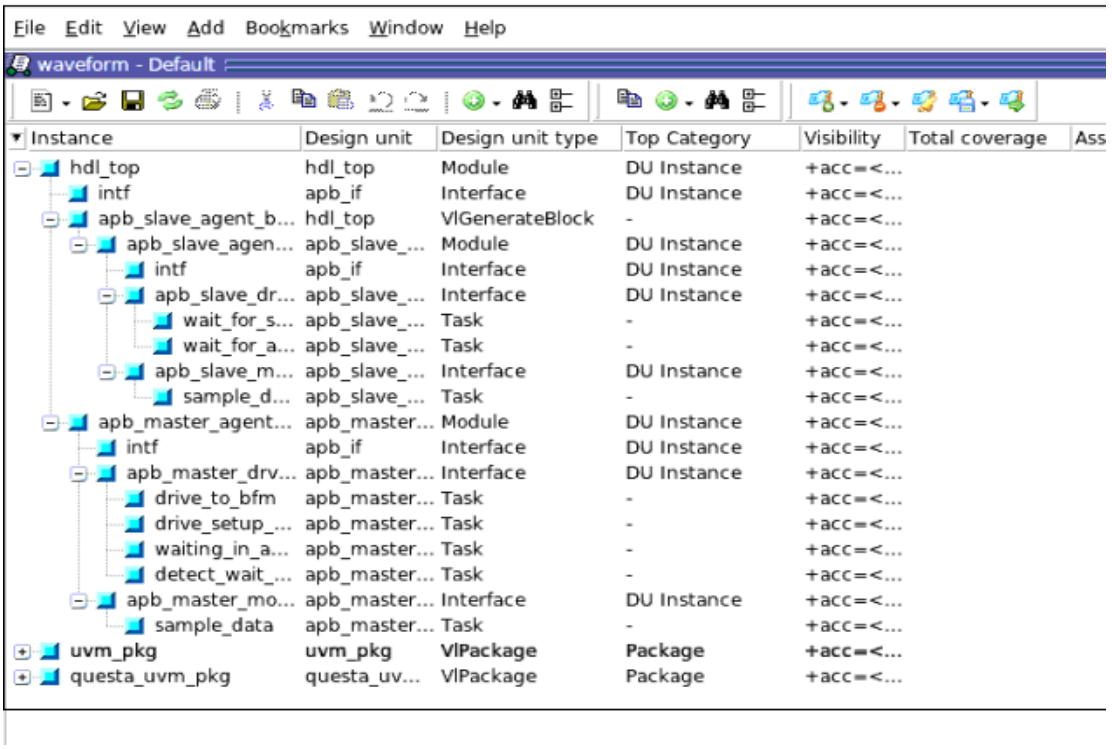


Fig 3.2 Questasim WLF window

5. Right-click on intf and select Add Wave as shown in the image 3.3 to add the signals to the wave window

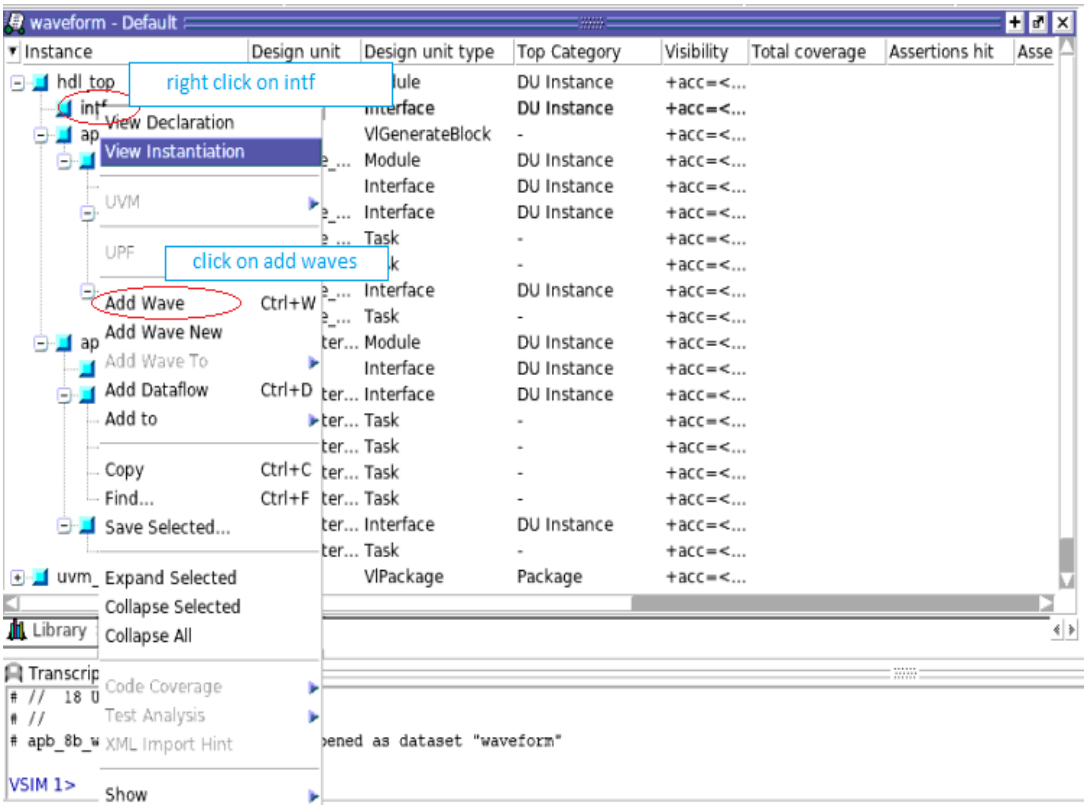


Fig 3.3 Screenshot of adding waves in wave window

6. After adding wave, click on Wave window as shown in the Fig 3.3

Fig 3.4 Wave window

7. Click as shown in the fig 3.4 to unlock the waveform window

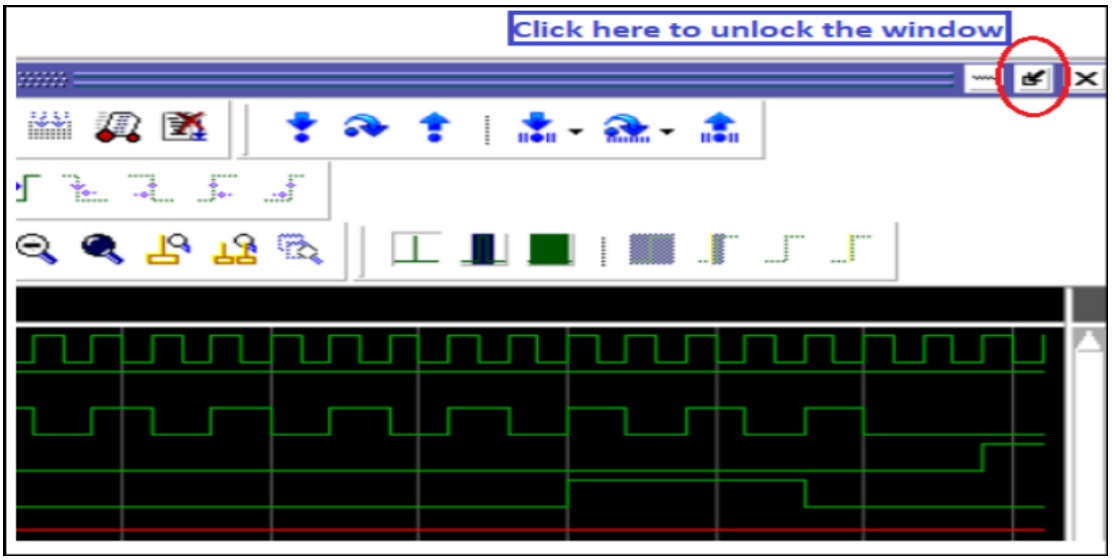


Fig 3.5 Screenshot of unlocking the wave window

8. You will be able to get a separate wave window as shown in the Fig 3.5

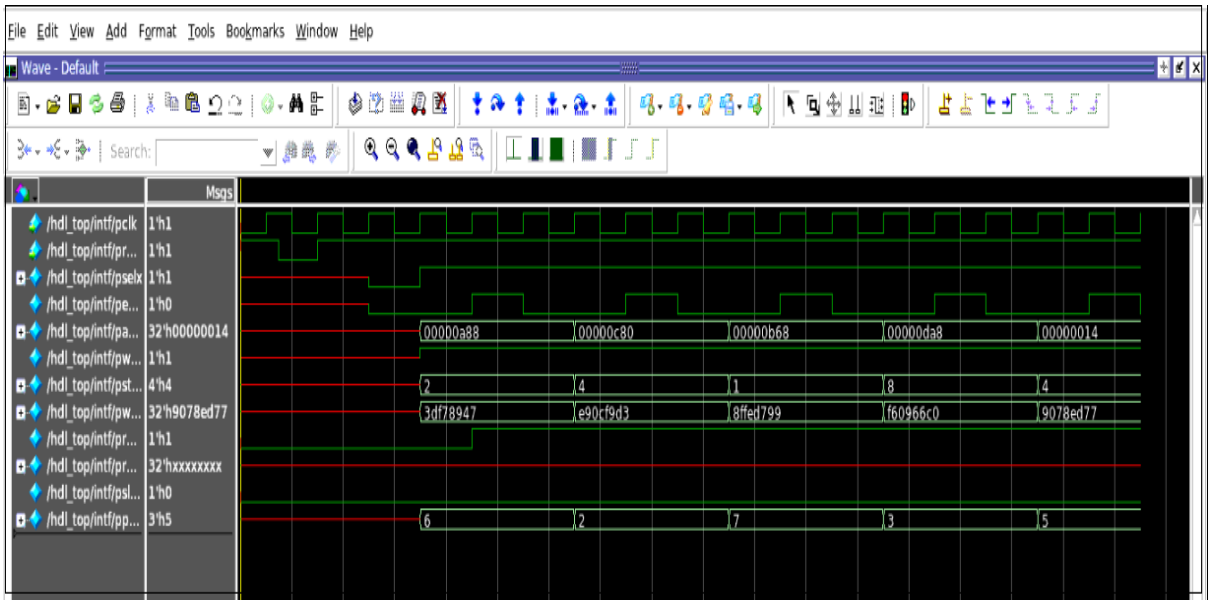


Fig 3.6 Screenshot of unlocked the wave window with signals

9. Click on the icon signal toggle leaf name marked in fig 3.6 to see the signals as shown

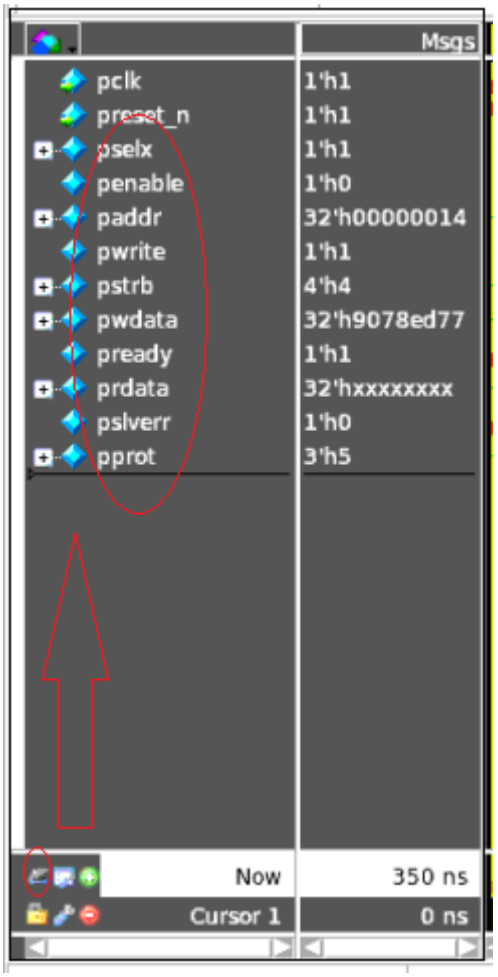


Fig 3.7 Screenshot showing way to see the name of the signals

10. For the analysis of waveform, go through the link below

    Waveform Viewer

3.2.3 Regression

1. To run regression for all test case

   make regression testlist_name=<regression_testlist_name.list>

   Ex: *make regression testlist_name=apb_regression.list*

   Note: You can find all the test case names in the path given below
   APB_avip/src/hvl_top/testlists/apb_regression.list

2. After regression, you can view the individual files as shown fig 3.7

   ls

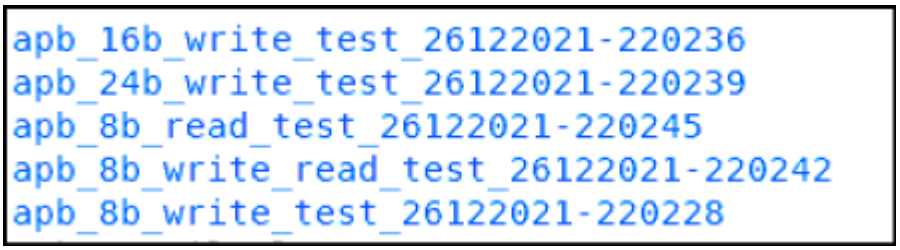


Fig 3.8 Files in questasim after the regression

3. To view the log files of individual test, select the interested test case file, go inside that directory

   Ex:Interested in the test case *apb_8b_write_test*

   Go inside the directory of interested testcase with the date

   apb_8b_write_test_26122021-220228

   Inside this directory, you will be able to find the log file of the interested test case

   apb_8b_write_test.log

   Path:

   apb_8b_write_test_26122021-220228/apb_8b_write_test.log

3.2.4 Coverage

1. To see coverage

a. After simulating

For the individual test, use the command firefox

firefox apb_8b_write_test/html_cov_report/index.html &

Ex: *firefox apb_8b_write_test/html_cov_report/index.html &*

Note: The command to see the coverage will be displayed in the simulation report along with the name of the simulated test

b. **After the regression,**

- To view the coverages of all test cases, type the below command

  firefox merged_cov_html_report/index.html &

  Note: The command to see the coverage will be displayed in the simulation report along with the name of the simulated test

- To view the coverage for individual test case

  See the list of files generated after regression, which is shown in fig 3.7.

  Select the interested test case file, go inside that directory

Ex:

Interested in the test case *apb_8b_write_test*

Go inside the directory of interested testcase with the date

ADD

Inside this directory, you will be able to find the html coverage file of the interested test case

html_cov_report/

Inside it would be the html file

covsummary.html

Command to view coverage report for the above test case will be

ADD

2. The coverage report window appears as shown in fig 3.8

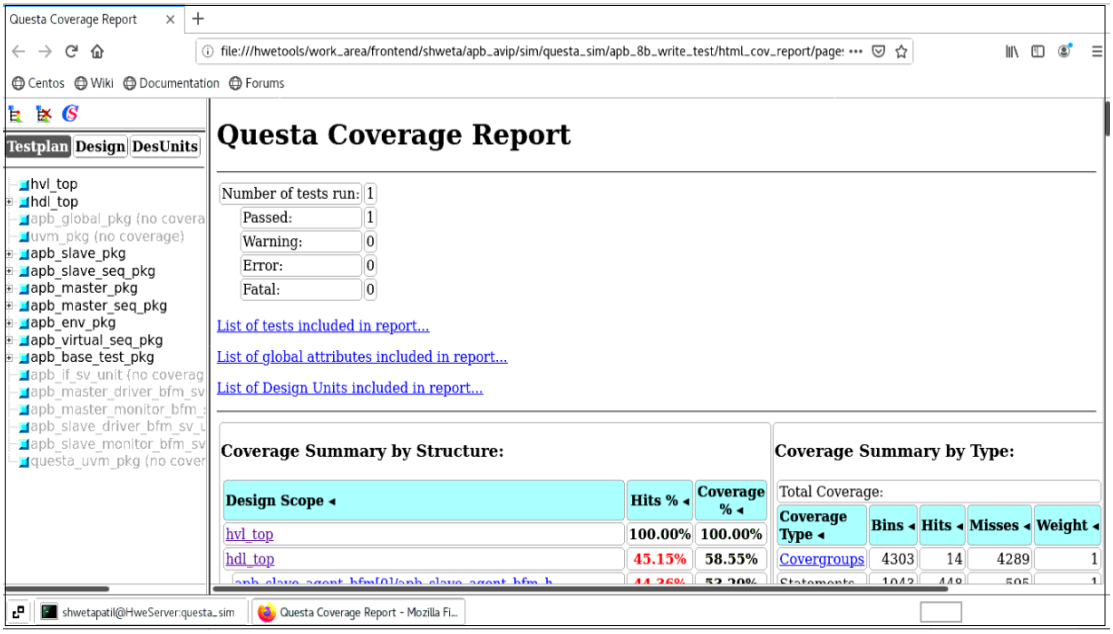
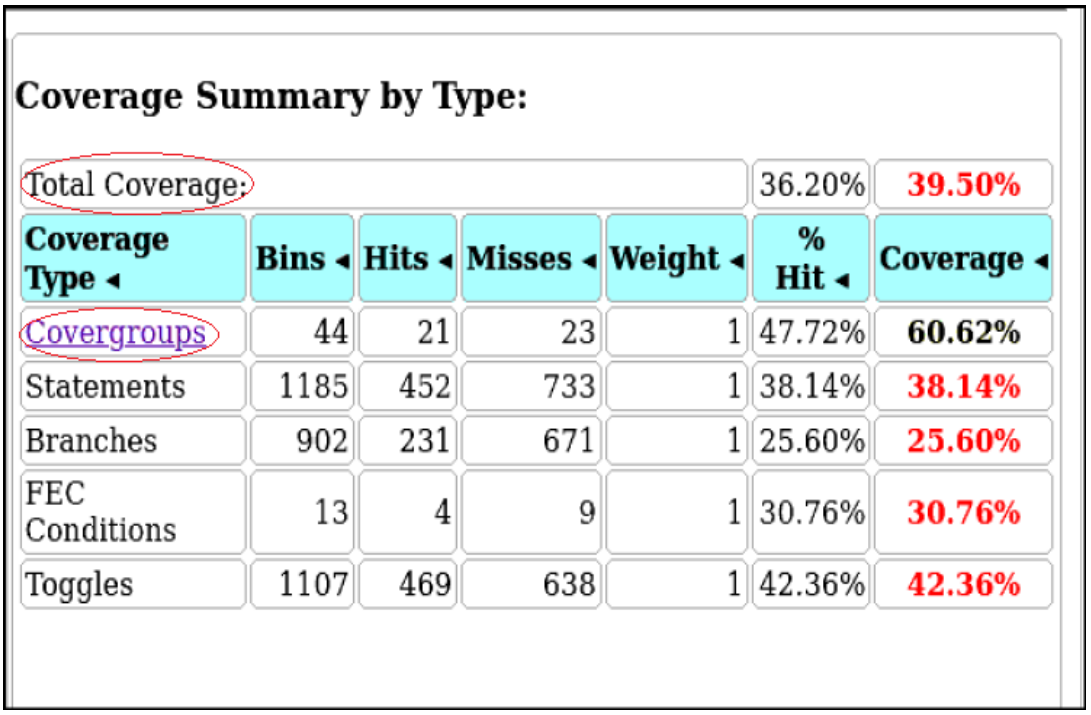


Fig 3.9 Coverage Report

3. Scroll down to the coverage summary by type and click on covergroups shown in fig 3.9.

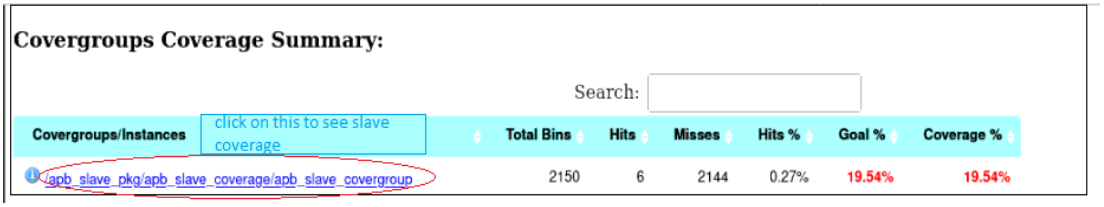


Coverage Summary by Type:

Total Coverage:					36.20%	39.50%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	44	21	23	1	47.72%	60.62%
Statements	1185	452	733	1	38.14%	38.14%
Branches	902	231	671	1	25.60%	25.60%
FEC Conditions	13	4	9	1	30.76%	30.76%
Toggles	1107	469	638	1	42.36%	42.36%

3.10 Screenshot of opening covergroups

4. After opening covergroup you will be able to see the summary.click on as shown in the fig 3.10 to slave covergroup



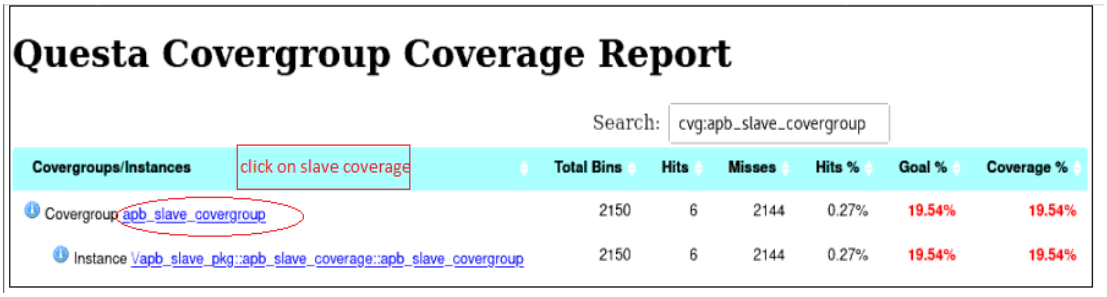
Covergroups Coverage Summary:

Search:

Covergroups/Instances	click on this to see slave coverage	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
/apb_slave_pkg/apb_slave_coverage/apb_slave_covergroup		2150	6	2144	0.27%	19.54%	19.54%

3.11 Screenshot of opening slave covergroup

5. If clicked on slave covergroup, further it opens to another window, again click on the slave covergroup as shown in fig 3.11



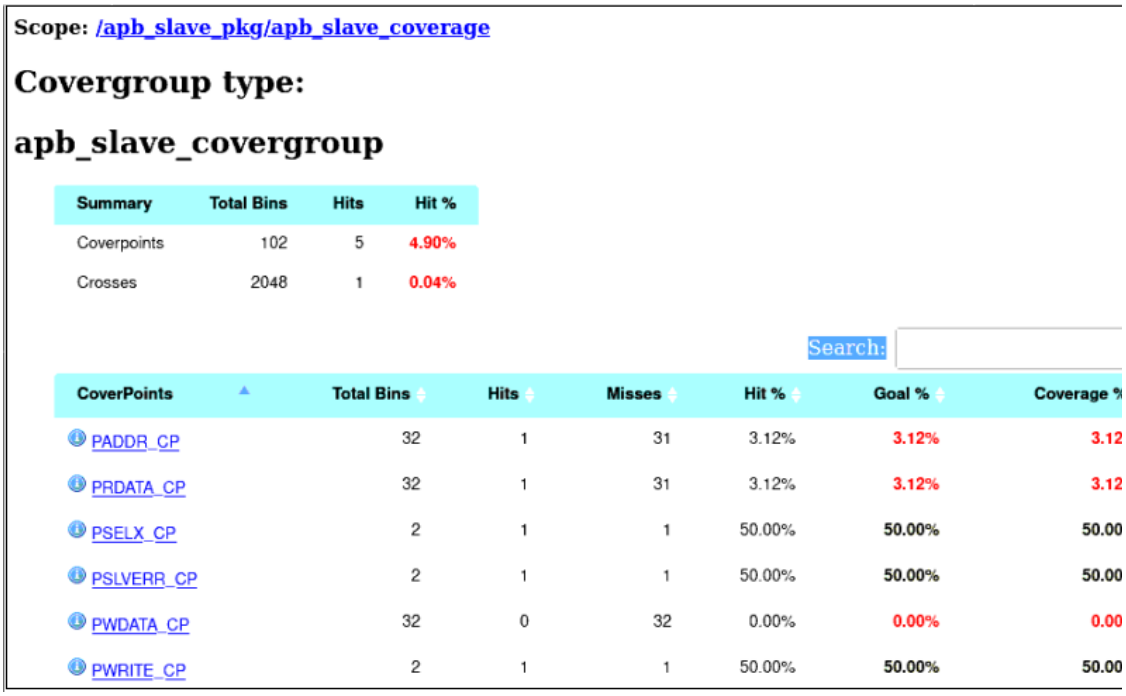
Questa Covergroup Coverage Report

Search: cvg:apb_slave_covergroup

Covergroups/Instances	click on slave coverage	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
Covergroup apb_slave_covergroup		2150	6	2144	0.27%	19.54%	19.54%
Instance \/apb_slave_pkg::apb_slave_coverage::apb_slave_covergroup		2150	6	2144	0.27%	19.54%	19.54%

3.12 Shows way to open slave covergroup coverage report

6. Further, you will be able to see coverpoints and crosses as shown in fig 3.12



Scope: /apb_slave_pkg/apb_slave_coverage

Covergroup type:

apb_slave_covergroup

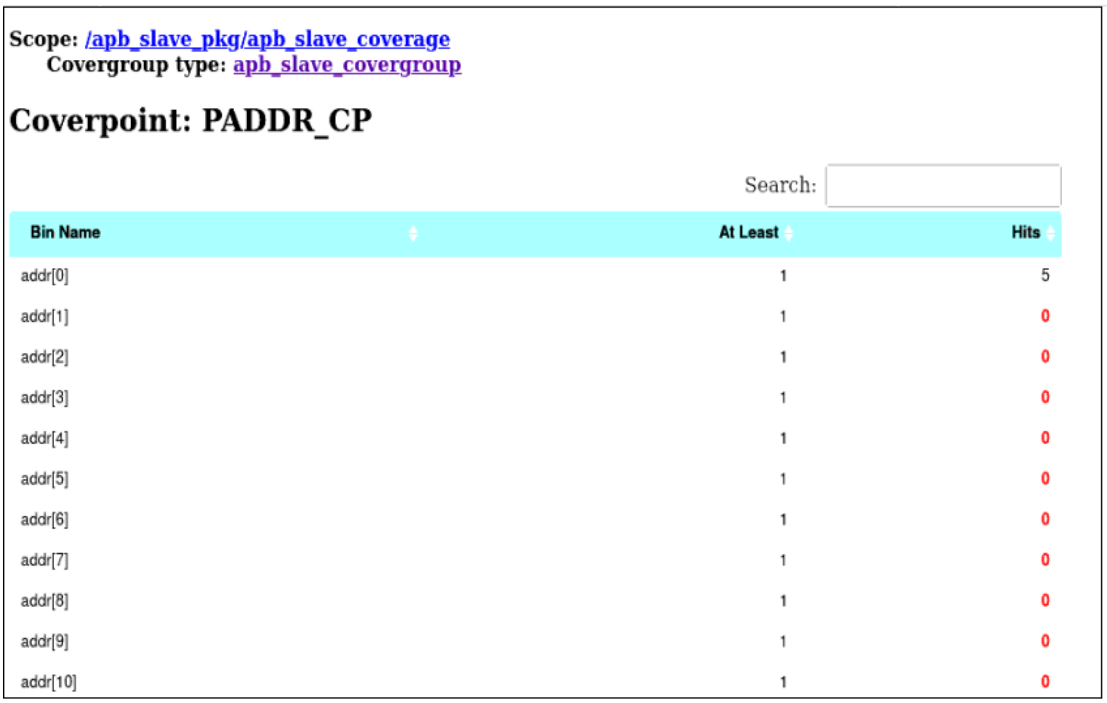
Summary	Total Bins	Hits	Hit %
Coverpoints	102	5	4.90%
Crosses	2048	1	0.04%

Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_CP	32	1	31	3.12%	3.12%	3.12
PRDATA_CP	32	1	31	3.12%	3.12%	3.12
PSELX_CP	2	1	1	50.00%	50.00%	50.00
PSLVERR_CP	2	1	1	50.00%	50.00%	50.00
PWDATA_CP	32	0	32	0.00%	0.00%	0.00
PWRITE_CP	2	1	1	50.00%	50.00%	50.00

Fig 3.13 Screenshot of Coverpints and cross coverpoints

7. Click on individual coverpoints and crosses to see the bins hit, here PADDR_CP is individual coverpoint in fig 3.13



Scope: /apb_slave_pkg/apb_slave_coverage

Covergroup type: apb_slave_covergroup

Coverpoint: PADDR_CP

Search:

Bin Name	At Least	Hits
addr[0]	1	5
addr[1]	1	0
addr[2]	1	0
addr[3]	1	0
addr[4]	1	0
addr[5]	1	0
addr[6]	1	0
addr[7]	1	0
addr[8]	1	0
addr[9]	1	0
addr[10]	1	0

Fig. 3.14 PADDR_CP coverpoint report

8. For the analysis of coverage report, click on the link *Coverage Debug*

3.3 Cadence

1. Change the directory to questasim directory where the makefile is present

   Path for the mentioned directory is APB_avip/sim/cadence_sim

   Note: To Compile, simulate, regression and for coverage you must be in the specified path i.e, *apb_avip/sim/cadence_sim*

2. To view the usage for running test cases, type the command

   make

   Fig 3. shows the usage to compile, simulate, and regression

```
------------------------------------------------------------------
------------------------ Usage -----------------------------------

make target <options> <variable>=<value>

To compile use:
make compile

To simulate use:
make simulate test=<test_name> uvm_verbosity=<VERBOSITY_LEVEL>

Example::
make simulate test=base_test uvm_verbosity=UVM_HIGH

------------------------------------------------------------------
------------------------------------------------------------------
```

Fig 3.15 Usage of make command in cadence

3.3.1 Compilation

1. Use the following command to compile

   make compile

2. Open the log file apb_compile.log to view the compiled files
   vim apb_compile.log

3.3.2 Simulation

1. After compilation, use the following command to simulate individual test cases

   make simulate test=<test_name> uvm_verbosity=<VERBOSITY_LEVEL>

   Example:

Note: You can find all the test case names in the path given below
apb_avip/src/hvl_top/testlists/apb_simple_fd_regression.list

2. To view the log file
   gvim <test_name>/<test_name>.log

   Ex: *gvim apb_8b_write_test/apb_8b_write_test.log*

   Note: The path for the log file will be displayed in the simulation report along with the name of the simulated test

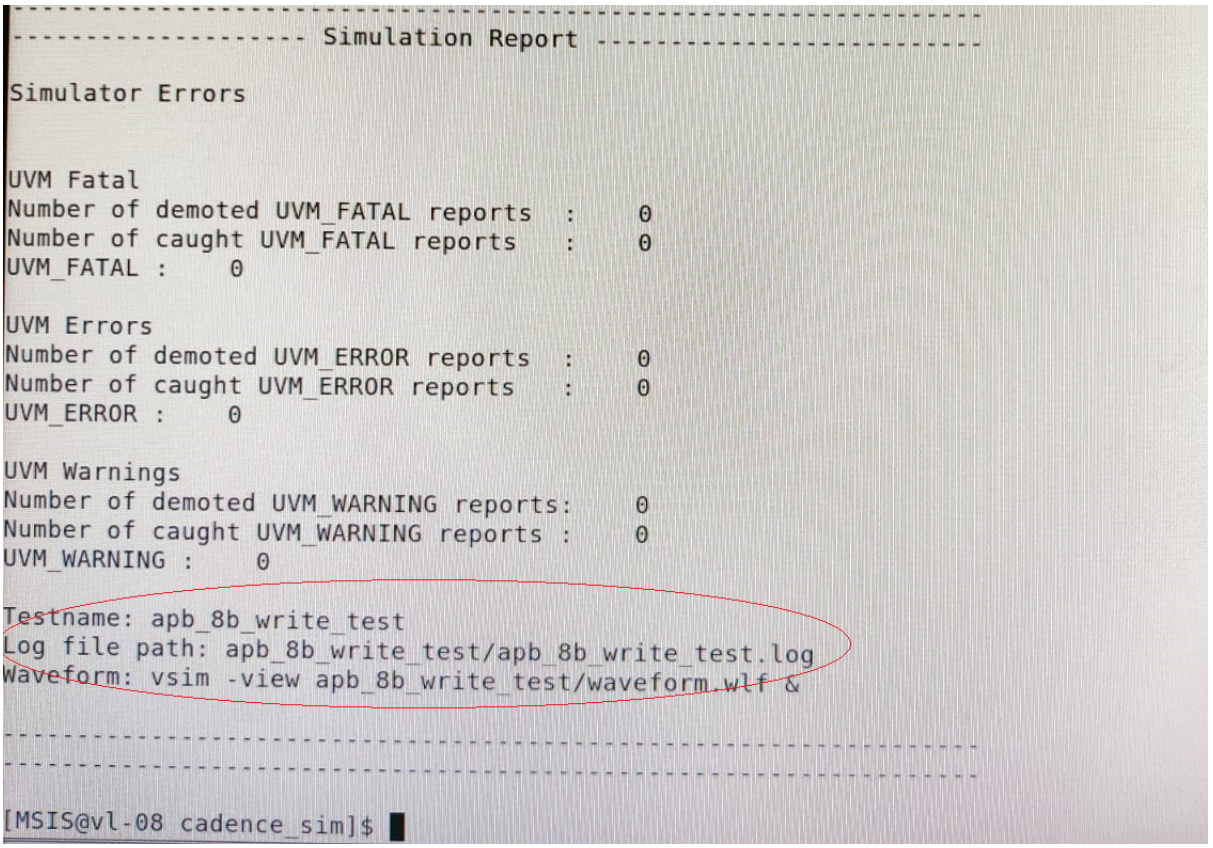


Fig 3.16 Simulation report in cadence

3. To view waveform

   simvision waves.shm/

   Ex: *simvision waves.shm/*

   Note: The command to view the waveform will be displayed in the simulation report along with the name of the simulated test as waveform shown in the fig 3.15

3.3.3 Coverage

Command to see the coverage after simulation : *imc -load cov_work/scope/test*

Chapter 4

Debug Tips

As design complexity continues to increase, which is contributing to new challenges in verification and debugging. Fortunately, new solutions and methodologies (such as UVM) have emerged to address growing design complexity. Yet, even with the productivity gains that can be achieved with the adoption of UVM, newer debugging challenges specifically related to UVM need to be addressed.

Here **apb_8b_write_test** has been used as an example test case in order to show the below debugging flow of the APB protocol and all the info's have been runned using **UVM_HIGH** verbosity

4.1 APB Debugging Flow

Initially, open with a log file which is inside the test folder that has been run and then follow the below procedure in order to have a debug flow.

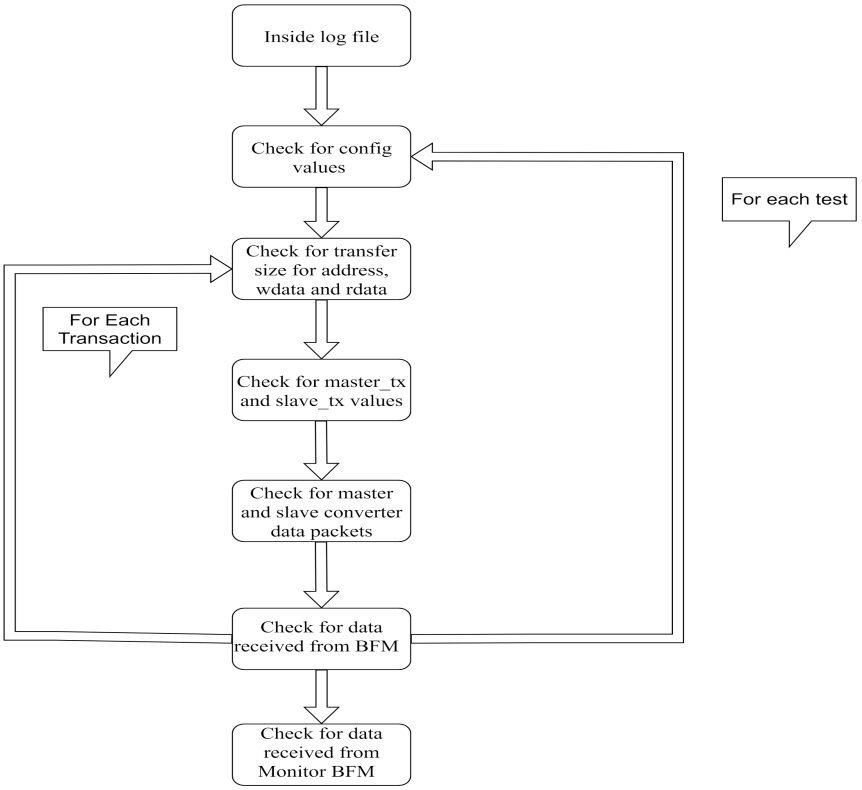


Fig 4.1 Debugging flow

4.1.1 Check for Configuration Values

At this stage, the user is trying to check for all the values related to master agent, slave agent and environment configurations which have been generated from the test.

For more information on Configurations please visit the following link:
Configuration Doc

4.1.1(a) Master agent configurations

Master agent configurations Includes

Table 4.1 master configurations

Configurations	conditions for the configurations
No of Slaves	which should not equal to zero
has_coverage	Which indicates the coverage connection
master_min_addr_range_array[0]	Which indicate the minimum address range for master
master_max_addr_range_array[0]	Which indicate the maximum address range for master

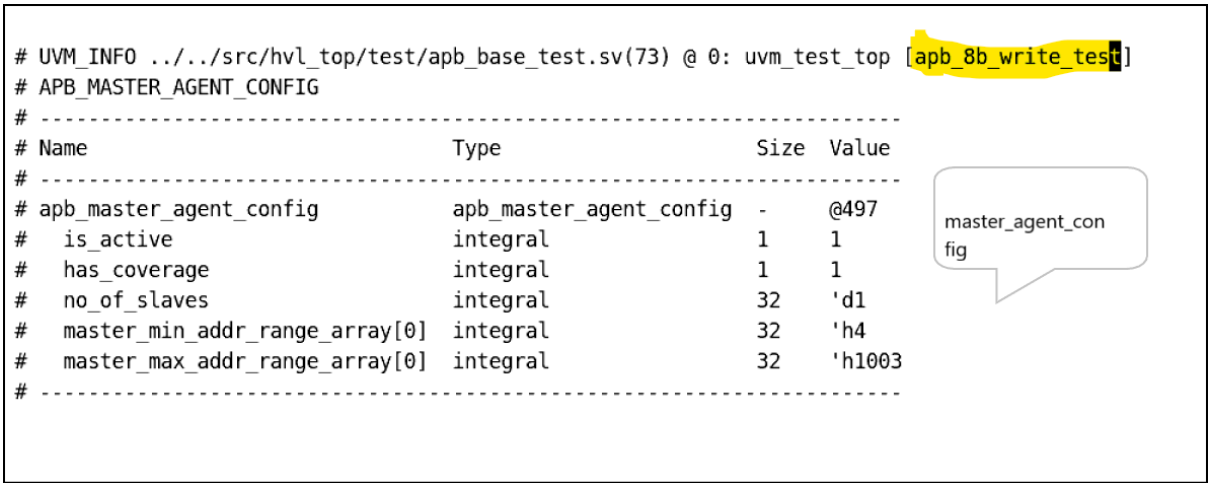


Fig 4.2 master_agent_config values

Figure 4.2 shows the different config values that have been set in master agent config class

4.1.1(b) Slave agent configurations

Slave agent configurations Includes

Table 4.2 slave configurations

Configurations	conditions for the configurations
has_coverage	Which indicates the coverage connection
Slave_id	Tells which slave is selected
max_address	Which indicates the maximum address for slave
min_address	Which indicates the minimum address for slave

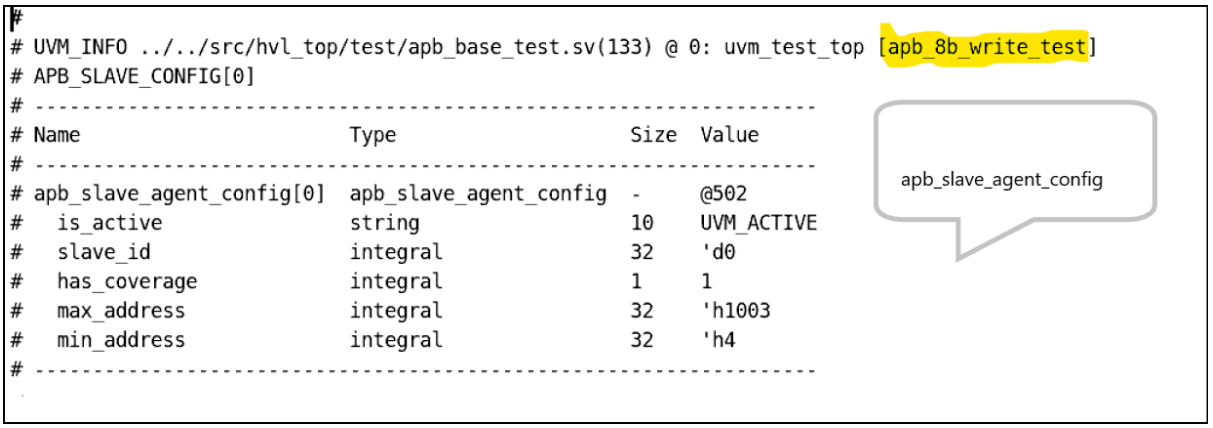


Fig 4.3 slave_agent_config values

Figure 4.3 shows the different config values that has been set in slave agent config class

4.1.1(c) Environment configuration

Environment configuration includes

Table 4.3 environment configurations

Configurations	conditions for the configurations
has_scoreboard	which tells how many scoreboards are connected to env. Which has to be at least 1
has_virtual_seqr	which tells how many virtual seqr are connected to env Which has to be at least 1
No_of_slaves	Tells how many slaves are connected Which shouldn't be 0

```
#
# UVM_INFO ../../src/hvl_top/test/apb_base_test.sv(78) @ 0: uvm_test_top [apb_8b_write_test]
# APB_ENV_CONFIG
# ------------------------------------------------
# Name                Type            Size  Value
# ------------------------------------------------
# apb_env_cfg_h       apb_env_config  -     @496
#   has_scoreboard    integral        1     1
#   has_virtual_seqr  integral        1     1
#   no_of_slaves      integral        32    'd1
# ------------------------------------------------
```

Fig 4.4 env_config values

Figure 4.4 shows the different config values that has been set in env config class

4.1.2 Check for transfer size

APB transfers the data based on pstrobe signal. Each pstrobe lane can transfer a byte data. The pstrobe lane becomes high based on transfer size declaration only.

```
----------------------------------------------------------------------------
Name                          Type           Size  Value
----------------------------------------------------------------------------
req                           apb_master_tx  -     @1119
  pselx                       string         7     SLAVE_0
  paddr                       integral       32    'ha88
  pwrite                      string         5     WRITE
  pwdata                      integral       32    'h3df78947
  transfer_size               string         5     BIT_8
  pstrb                       integral       4     'b10
  pprot                       string         25    PRIVILEGED_NONSECURE_DATA
  prdata                      integral       32    'h0
  pslverr                     string         8     NO_ERROR
  no_of_wait_states_detected  integral       32    'd0
----------------------------------------------------------------------------
```

Fig 4.5 Transfer size for pwdata

4.1.3 Check for transaction values

Once the config values and size of the transfers are correct then check for the data to be transmitted from master_tx class or from slave_tx class

Initially check for the idle state of the transaction.(Ex: In this case pselx = 0 and penable =). Once the present is high based on the pclk edge,when the pselx becomes high the data will be sampled on the same clock edge.

```
# UVM_INFO ../../src/hvl_top/master//apb_master_tx.sv(209) @ 50: uvm_test_top.apb_env.apb_master_agent.apb_master_seqr_h@
# @apb_master_8b_seq_h.req [apb_master_tx] MASTER-TX-pstrb[3]=0
# ---------------------------------------------------------------------------
# Name                          Type          Size  Value
# ---------------------------------------------------------------------------
# req                           apb_master_tx -     @1119
#   pselx                       string        7     SLAVE_0
#   paddr                       integral      32    'ha88
#   pwrite                      string        5     WRITE
#   pwdata                      integral      32    'h3df78947
#   transfer_size               string        5     BIT_8
#   pstrb                       integral      4     'b10
#   pprot                       string        25    PRIVILEGED_NONSECURE_DATA
#   prdata                      integral      32    'h0
#   pslverr                     string        8     NO_ERROR
#   no_of_wait_states_detected  integral      32    'd0
# ---------------------------------------------------------------------------
```

Fig 4.6 master_tx values

```
# ---------------------------------------------------------------
# Name                  Type          Size  Value
# ---------------------------------------------------------------
# req                   apb_slave_tx  -     @1317
#   psel                integral      1     'd0
#   paddr               integral      32    'h0
#   pwrite              string        4     READ
#   pwdata              integral      32    'h0
#   pready              integral      1     'd0
#   prdata              integral      32    'h342ee465
#   pslverr             string        8     NO_ERROR
#   pprot               string        21    NORMAL_NONSECURE_DATA
#   no_of_wait_states   integral      3     'd2
#   choose_packet_data  integral      1     'd0
# ---------------------------------------------------------------
```

Fig 4.7 slave_tx values

Figure 4.6 and 4.7 shows the transaction data related to the master and slaves side.

4.1.4 Check for master and slave converter data packets

In the master and slave converter class the data coming from the req will convert into struct packet in from class and once the data driving and sampling done the data can be revert back to req using to_class method.

```
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(88) @ 340:
  reporter [apb_master_seq_item_conv] ----------------------------------------------
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(92) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pprot =  101
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(96) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pselx =  0000000000000001
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(99) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pwrite =  1
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(102) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize paddr =  14
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(105) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pwdata =  9078ed77
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(108) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pwdta =  0100
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(112) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize pslverr =  0
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(115) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize prdata =  0
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(118) @ 340:
  reporter [apb_master_seq_item_conv_class] After randomize no_of_wait_states =0
# UVM_INFO ../../src/hvl_top/master//apb_master_seq_item_converter.sv(120) @ 340:
  reporter [apb_master_seq_item_conv] ----------------------------------------------
```

Fig 4.8 converted data of master req

```
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(93) @ 340:
# reporter [apb_seq_item_conv_to_class] --
# ---------------------SLAVE_SEQ_ITEM_CONVERTER_TO_CLASS--------------------
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(96) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the paddr=14
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(100) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the pwdata=9078ed77
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(104) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the psel=1
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(108) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the pprot=5b
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(112) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the pslverr=0
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(116) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the pwrite=1
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(119) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the prdata=0
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(122) @ 340:
  reporter [apb_seq_item_conv_class] After randomizing the no_of_wait_states=0
# UVM_INFO ../../src/hvl_top/slave/apb_slave_seq_item_converter.sv(124) @ 340:
  reporter [apb_seq_item_conv_to_class] --
# ------------------------------EOP-------------------------------------
```

Fig 4.9 converted data of slave req

4.1.5 Check for data received from BFM

Once the data has been randomized and sent to master or slave BFMs. The master driver BFM will drive paddr,pwrite,pstrobe,pwdtata signals and samples the prdata,pslverr,pready depending on configurations of master and similarly slave driver BFM will drive the prdata,pready,pslverr signal and samples the paddr,psel,pwrite,pwdata depending on configurations of slave.

The master driver BFM will print both the all the signals which has been driven by the master and sampled data master and similarly slave driver BFM will print all the signal which has been driven by the slave and sampled data. At the end both the master BFM and slave BFM data has to be the same.

```
# UVM_INFO ../../src/hdl_top/master_agent_bfm/apb_master_driver_bfm.sv(83) @ 290: reporter [APB_MASTER_DRIVER_BFM] data_packet=
# '{pwrite:1, pslverr:0, pprot:5, pselx:1, pstrb:4, prdata:0, paddr:20, pwdata:2423844215, no_of_wait_states:0}
```

Fig 4.10 master bfm_struct data

```
# UVM_INFO ../../src/hvl_top/slave/apb_slave_driver_proxy.sv(210) @ 350: uvm_test_top.apb_env.apb_slave_agent_h[0].apb_slave_drv_proxy_h
# [DEBUG_NA] AFTER PSLVERR_CHECK_1 struct ::
 '{pwrite:1, pslverr:0, pprot:5, pselx:1, pstrb:4, prdata:0, paddr:20, pwdata:2423844215, no_of_wait_states:0}
```

Fig 4.11 slave bfm_struct data

The fig 4.10 and 4.11 shows the data with respect to pwrite,pradata,pwdata,pprot,pslverr,pstrb signals of both master and slave end before converting back to req using to_class converter.

```
# UVM_INFO ../../src/hvl_top/master/apb_master_driver_proxy.sv(109) @ 110: uvm_test_top.apb_env.apb_master_agent.apb_master_drv_proxy_h [apb_master_driv
er_proxy] REQ-MASTER_TX
#  ------------------------------------------------------------------------
# Name                          Type           Size  Value
# ------------------------------------------------------------------------
# req                           apb_master_tx  -     @1225
#   pselx                       string         7     SLAVE_0
#   paddr                       integral       32    'hc80
#   pwrite                      string         5     WRITE
#   pwdata                      integral       32    'he90cf9d3
#   transfer_size               string         5     BIT_8
#   pstrb                       integral       4     'b100
#   pprot                       string         21    NORMAL_NONSECURE_DATA
#   prdata                      integral       32    'h0
#   pslverr                     string         8     NO_ERROR
#   no_of_wait_states_detected  integral       32    'd0
# ------------------------------------------------------------------------
```

Fig 4.12 master_driver_bfm values

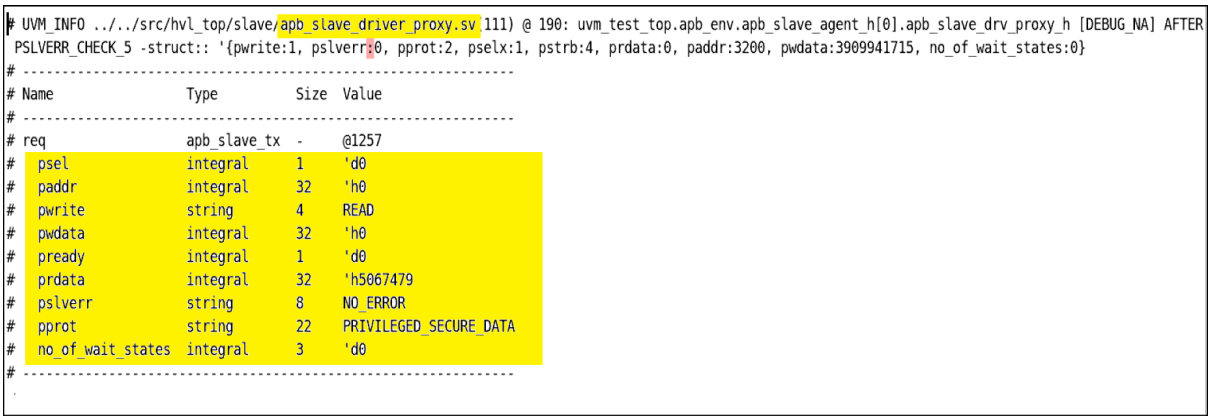


Fig 4.13 slave_driver_bfm values

Fig 4.13 and 4.14 shows the psel, paddr, pwrite, pwdata, pready, prdata, pslverr, pprot values from master and slave bfm driver after converting back to req.

4.1.6 Check for data received from monitor BFM

Once the data has been driven or sampled monitor will capture the data and it will print the driven and sampled data in the req form or transaction level

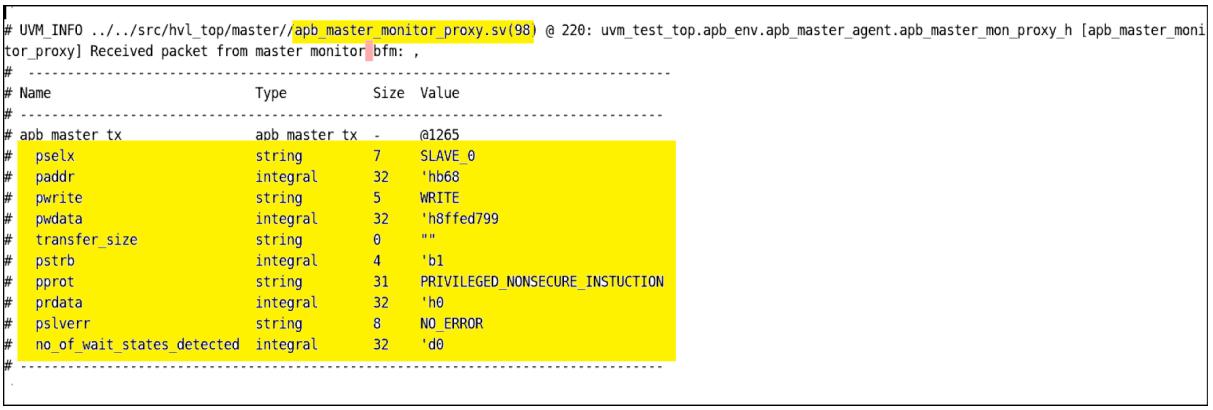


Fig 4.14 master_monitor values

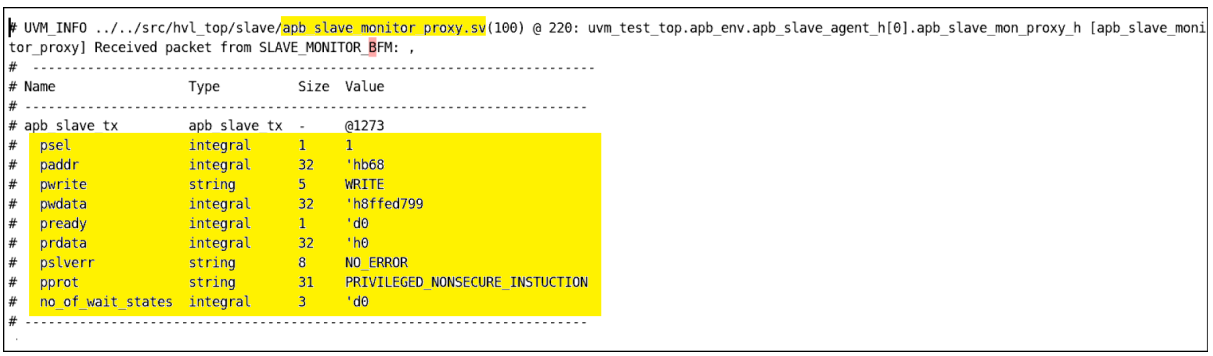


Fig 4.15 slave_monitor values

4.2 Scoreboard Checks

And finally we have scoreboard checks which basically compares the paddr, pwrite, prdata, pwdata, pprot data of master with the slave side

```
# ------------------------------------------------SCOREBOARD COMPARISIONS------------------------------------------------
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(162) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard]
# apb_pwdata from master and slave is equal
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(164) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [SB_PWDATA_MATCHED]
# Master PWDATA = 'hf60966c0 and Slave PWDATA = 'hf60966c0
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(188) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard]
# apb_paddr from master and slave is equal
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(190) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [SB_PADDR_MATCH]
# Master PADDR = 'hf60966c0 and Slave PADDR = 'hf60966c0
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(216) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard]
# apb_pwrite from master and slave is equal
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(218) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [SB_PWRITE_MATCH]
# Master PWRITE = 'h1 and Slave PWRITE = 'h1
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(246) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard]
# apb_prdata from master and slave is equal
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(248) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [SB_PRDATA_MATCHED]
# Master PRDATA = 'h0 and Slave PRDATA = 'h0
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(260) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard]
# apb_prdata from master and slave is equal
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(262) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [SB_PPROT_MATCHED]
# Master PPROT = 'h3 and Slave PPROT = 'h3
# UVM_INFO ../../src/hvl_top/env/apb_scoreboard.sv(302) @ 280: uvm_test_top.apb_env.apb_scoreboard_h [apb_scoreboard] --
# ------------------------------------------------END OF SCOREBOARD COMPARISIONS-----------------------------------------
```

Fig 4.16 scoreboard_checks

4.3 Coverage Debug

Coverage is a metric which basically tells how much percentage of verification has been done to the dut.

Go to the log file, Here it will get you the complete master and slave coverage for the particular test we are running.

```
# UVM_INFO ../../src/hvl_top/master//apb_master_coverage.sv(148) @ 350:
# uvm_test_top.apb_env.apb_master_agent.apb_master_cov_h [apb_master_coverage] APB Master Agent Coverage = 58.75 %
```

Fig 4.17 coverage for master

```
UVM_INFO ../../src/hvl_top/slave/apb_slave_coverage.sv(118) @ 350:
uvm_test_top.apb_env.apb_slave_agent_h[0].apb_slave_cov_h [apb_slave_coverage] Slave Agent Coverage = 62.50 %
```

Fig 4.18 coverage for slave

For individual bins checking goto the below html file.
firefox apb_8b_write_test/html_cov_report/index.html &

Inside that check for covergroups in the coverage summary then check for the instance created for master and slave coverage

Covergroups Coverage Summary:

Search:

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
/apb_master_pkg/apb_master_coverage/apb_master_covergroup	34	15	19	44.11%	58.74%	58.75%
work.apb_master_pkg::apb_master_coverage/apb_master_covergroup	34	15	19	44.11%	58.74%	58.75%
/apb_slave_pkg/apb_slave_coverage/apb_slave_covergroup	10	6	4	60.00%	62.50%	62.50%
work.apb_slave_pkg::apb_slave_coverage/apb_slave_covergroup	10	6	4	60.00%	62.50%	62.50%

Fig 4.19 master and slave coverage

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
Covergroup apb_master_covergroup	34	15	19	44.11%	58.74%	58.75%
Instance \/apb_master_pkg::apb_master_c....verage::apb_master_covergroup	34	15	19	44.11%	58.74%	58.75%

Fig 4.20 instance of cover group

Then click on the master covergroup instance to check the individual bins which are hitted and missed. And here you can even check cross coverages between pwdata, prdata, paddr, pstrobe.

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_CP	1	1	0	100.00%	100.00%	100.00%
PPROT_CP	8	5	3	62.50%	62.50%	62.50%
PRDATA_CP	1	1	0	100.00%	100.00%	100.00%
PSEL_CP	1	1	0	100.00%	100.00%	100.00%
PSLVERR_CP	2	1	1	50.00%	50.00%	50.00%
PSTRB_CP	16	4	12	25.00%	25.00%	25.00%
PWDATA_CP	1	0	1	0.00%	0.00%	0.00%
PWRITE_CP	2	1	1	50.00%	50.00%	50.00%

Fig 4.21 master_coverage coverpoint

Figure 4.21 shows all the coverpoints included in master coverage

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_CP_X_PRDATA_CP	1	1	0	100.00%	100.00%	100.00%
PADDR_CP_X_PWDATA_CP	1	0	1	0.00%	0.00%	0.00%

Fig 4.22 master_coverage crosses coverpoints

Figure 4.22 shows all the cross coverpoints included in master coverage

If you click on the slave covergroup instance to check the individual bins which are hitted and missed. And here you can even check cross coverages between pwdata , prdata, paddr, pstrobe.

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_CP	1	1	0	100.00%	100.00%	100.00%
PRDATA_CP	1	1	0	100.00%	100.00%	100.00%
PSELX_CP	1	1	0	100.00%	100.00%	100.00%
PSLVERR_CP	2	1	1	50.00%	50.00%	50.00%
PWDATA_CP	1	0	1	0.00%	0.00%	0.00%
PWRITE_CP	2	1	1	50.00%	50.00%	50.00%

Fig 4.23 slave_coverage coverpoint

Figure 4.23 shows all the coverpoints included in slave coverage

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_X_PRDATA_	1	1	0	100.00%	100.00%	100.00%
PADDR_X_PWDATA_	1	0	1	0.00%	0.00%	0.00%

Fig 4.24 slave_coverage crosses coverpoints

Figure 4.24 shows all the cross coverpoints included in slave coverage

4.4 Waveform Viewer

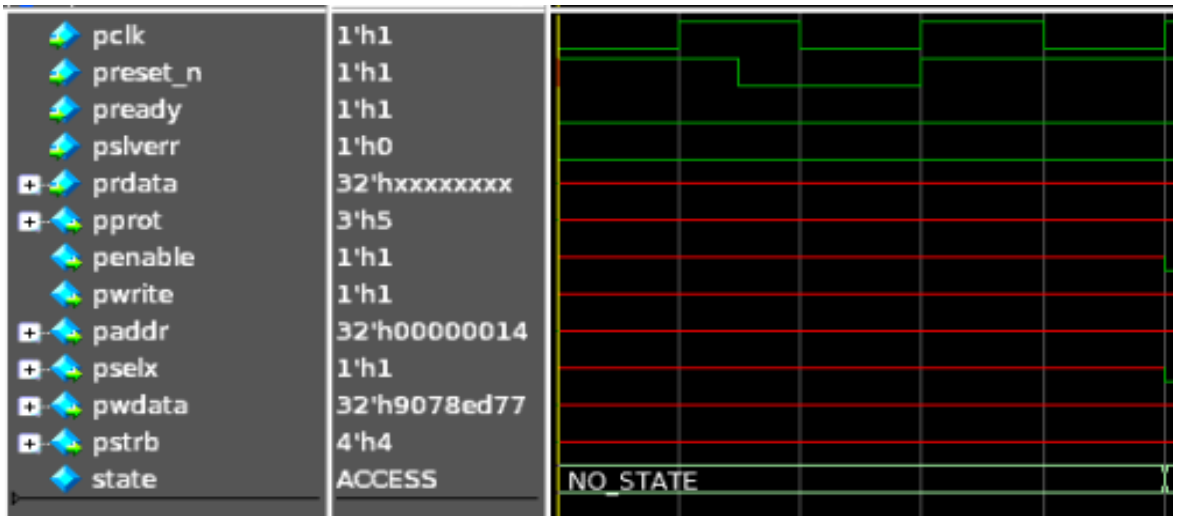


Fig 4.25 waveform for the 8 bit write when reset is low

1. In the waveform, initially check for the generation of the system clock(pclk), after every 10ns it will be toggled as shown in figure 4.25. Once the pclk is done check for the reset condition(Active low reset) if the reset is low the other signals such as pselx, penable, paddr, pstrb, pwrite, pwdata, pprot, prdata, pready, pslverr signals should be in unknown state.

2. Once the reset is high at the next posedge of pclk the psel and penable should come to idle state i.e., pselx = 0, penable = 0. The other signals paddr, pstrb, pwrite, pwdata, pprot, prdata, pready, pslverr should be in unknown state.

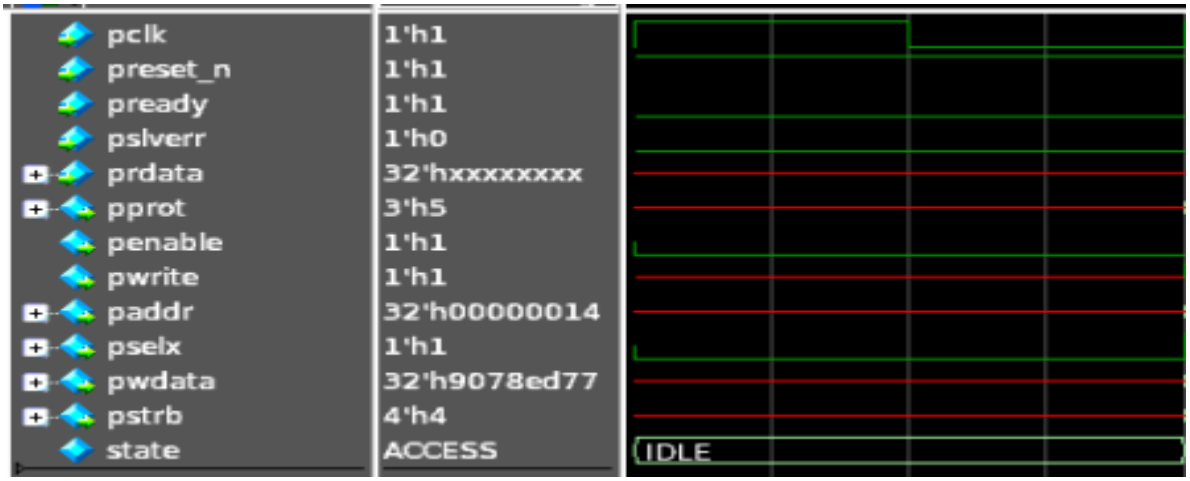


Fig 4.26 waveform for 8-bit write idle-state

3. After the idle phase is completed, pselx signal should select a slave and penable should be low which means APB is in SETUP phase i.e., pselx = SLAVE_NUMBER, penable = 0.
So, all the signals should be known data.

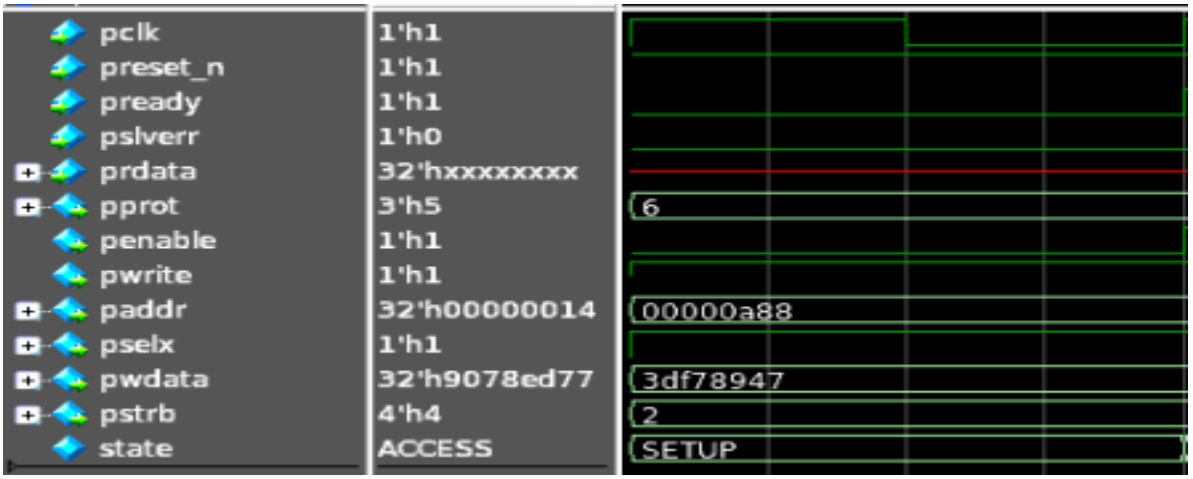


Fig 4.27 waveform for 8-bit write setup-phase

4. Now APB will be in the access phase i.e., pselx =SLAVE_NUMBER and penable =1, then check for pready if it is high the access state should end else it should enter wait state and the transaction is completed.

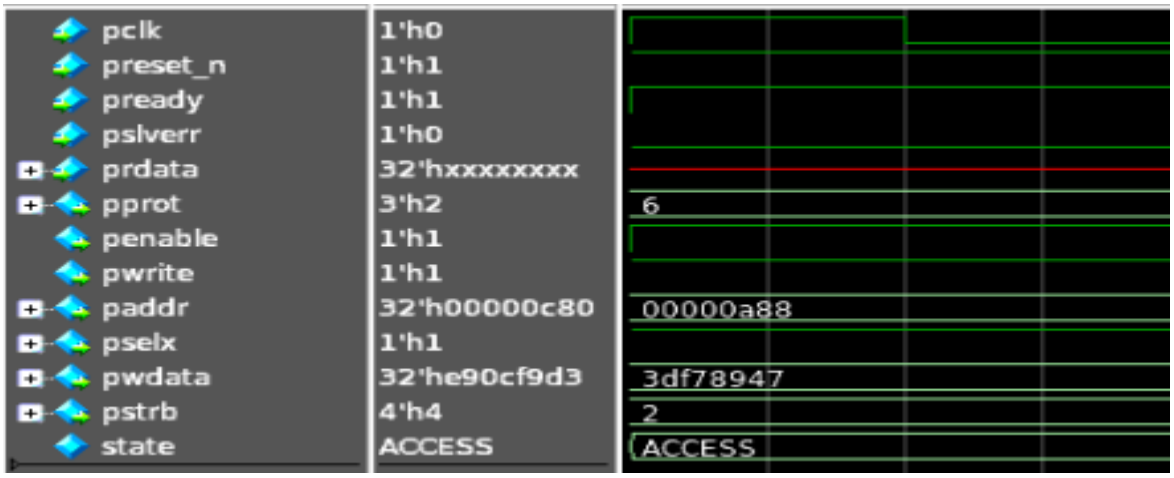


Fig 4.28 waveform for 8-bit write access-phase

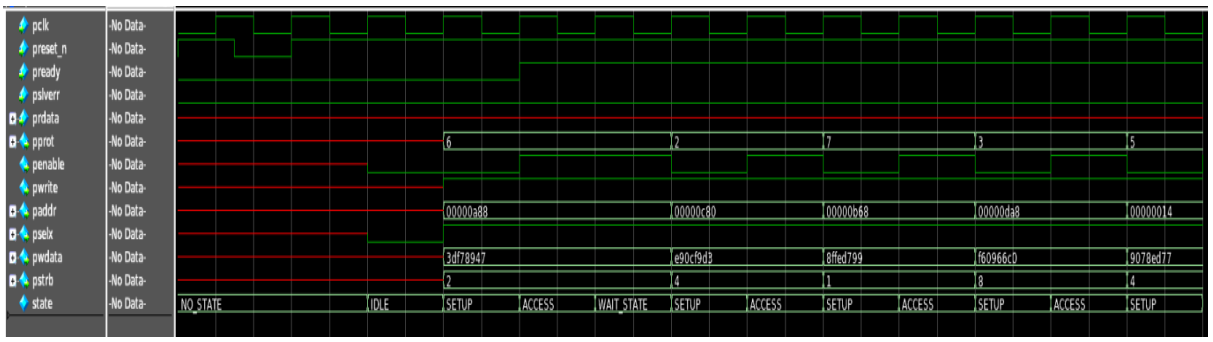


Fig 4.29 waveform for the 8 bit write transfer with repeat of 5 times

Figure 4.29 shows the waveform for 5 consecutive write transfers.

Chapter 5

References

https://github.com/git-guides/install-git

apb_avip_architectural_document