2021

PULPINO_SPI_MASTER_ SUBSYSTEM_VERIFICATIO N

USER GUIDE

Contents

List of Tables

List of Figures

Chapter 1

Introduction

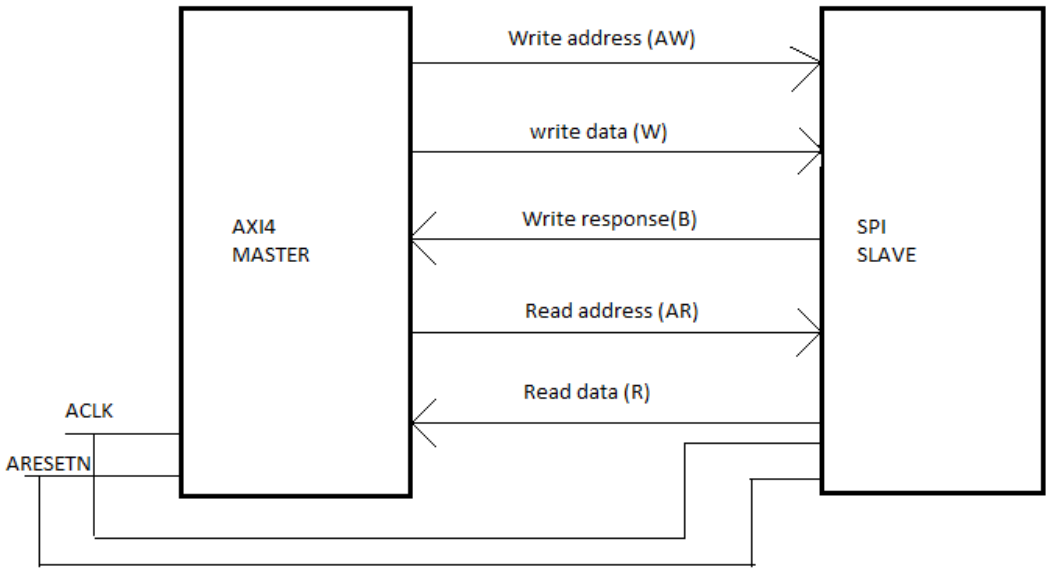


Fig: 1.1 Simple AXI4

Master avip can communicate with the slave avip via AXI4 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 Axi4 master avip

Axi4 master avip starts the test in the hvl top and starts the randomised sequences in base_test and will pass the awaddr,awdata and awid to Axi4 master_driver proxy using uvm sequencer and driver handshake. The Axi4 master driver bfm gets the awaddr,awdata and awid using inbound communication. The Axi4 master driver bfm sends the awaddr, awid and awdata an that is sampled in Axi4 master driver bfm and sends it back to the Axi4 master driver proxy. The Axi4 master monitor bfm samples the master_tx and slave_tx received from the Axi4 interface and sends it to the Axi4 master monitor proxy. The sampled data received by Axi4

master monitor bfm is sent to the Axi4 master scoreboard and Axi4 master coverage. Axi4 master scoreboard compares the driven data and sampled data. Axi master coverage is used to check the functional coverage of Axi4 master.

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

1.2 SPI slave avip

Spi slave avip starts the test in the hvl top and starts the randomised sequences in base_test and will pass the miso_data i.e., master_in_slave_out to spi slave_driver proxy using uvm sequencer and driver handshake. The spi slave driver bfm gets the miso_data using inbound communication. The spi slave driver bfm sends the miso_data that is sampled in spi slave driver bfm and sends it back to the spi slave driver proxy. The spi slave monitor bfm samples samples the mosi_data and miso_data received from the spi interface and sends it to the spi slave monitor proxy. The sampled data received by spi slave monitor bfm is sent to the spi slave scoreboard and spi slave coverage. Spi slave scoreboard compares the driven data and sampled data. Spi slave coverage is used to check the functional coverage of spi slave.

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

1.3 AXI4 Interface and Spi Interface :

APB and SPI interface has the following interface pin level signals :

1. Aclk
2. Aresetn
3. Awid
4. Awaddr
5. Awlen
6. Awsize
7. Awburst
8. Awlock
9. Awcache
10. Awprot
11. Awqos
12. Awregion
13. Awuser
14. Awvalid
15. Awready
16. Wdata
17. Wstrb
18. Wlast

19. Wiser
20. Valid
21. Wready
22. Bid
23. Bresp
24. Buser
25. Valid
26. bready
27. sclk
28. cs
29. cs1
30. cs2
31. cs3
32. mosi0
33. mosi1
34. mosi2
35. mosi3
36. miso0
37. miso1
38. miso2
39. miso3

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

1.4 verif_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 Axi4 interface
6. The SPI interface

- If you want to add any file in the project, please add the file or folder in the verif_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 verif_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 verif_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/dv/globals/
+incdir+../../src/dv/hvl_top/test/sequences/master_sequences/
+incdir+../../src/dv/hvl_top/test/sequences/slave_sequences/
+incdir+../../src/dv/hvl_top/test/sequences/master_sequences/reg_sequences/
+incdir+../../src/dv/hvl_top/test/virtual_sequences/
+incdir+../../src/dv/hvl_top/axi4_master_agent/
+incdir+../../src/dv/hvl_top/spi_slave_agent/
+incdir+../../src/dv/hvl_top/env/virtual_sequencer/
+incdir+../../src/dv/hvl_top/env/
+incdir+../../src/dv/hvl_top/test/
+incdir+../../src/dv/systemRDL
+incdir+../../src/dv/systemRDL/output
```

Fig: 1.2 Directories included in verif_compile.f file

b. Packages included:

```
../../src/dv/globals/axi4_globals_pkg.sv
../../src/dv/globals/spi_globals_pkg.sv
../../src/dv/globals/pulpino_spi_master_subsystem_global_pkg.sv
../../src/dv/hvl_top/axi4_master_agent/axi4_master_pkg.sv
../../src/dv/hvl_top/spi_slave_agent/spi_slave_pkg.sv
../../src/dv/systemRDL/output/spi_master_defines_pkg.svh
../../src/dv/systemRDL/output/spi_master_uvm_pkg.sv
../../src/dv/hvl_top/test/sequences/master_sequences/reg_sequences/axi4_master_reg_seq_pkg.sv
../../src/dv/hvl_top/test/sequences/master_sequences/axi4_master_seq_pkg.sv
../../src/dv/hvl_top/test/sequences/slave_sequences/spi_slave_seq_pkg.sv
../../src/dv/hvl_top/env/env_pkg.sv
../../src/dv/hvl_top/test/virtual_sequences/pulpino_spi_master_subsystem_virtual_seq_pkg.sv
../../src/dv/hvl_top/test/pulpino_spi_master_subsystem_test_pkg.sv
```

Fig: 1.3 Packages included in verif_compile.f file

C Static files included :

```
../../src/dv/hdl_top/axi4_interface/axi4_if.sv
../../src/dv/hdl_top/spi_interface/spi_if.sv
../../src/dv/hdl_top/axi4_master_agent_bfm/axi4_master_agent_bfm.sv
../../src/dv/hdl_top/axi4_master_agent_bfm/axi4_master_driver_bfm.sv
../../src/dv/hdl_top/axi4_master_agent_bfm/axi4_master_monitor_bfm.sv
../../src/dv/hdl_top/spi_slave_agent_bfm/spi_slave_agent_bfm.sv
../../src/dv/hdl_top/spi_slave_agent_bfm/spi_slave_driver_bfm.sv
../../src/dv/hdl_top/spi_slave_agent_bfm/spi_slave_monitor_bfm.sv
../../src/dv/hdl_top/hdl_top.sv
../../src/dv/hvl_top/hvl_top.sv
```

Fig: 1.4 static files included in verif_compile.f file

In Makefile, we include the verif_compile.f file to compile all the files included.
Command used : *irun -f verif_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 compile.log \
  -f ../rtl_compile.f \
  -f ../verif_compile.f
```

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

Chapter 2

Architecture

AXI4-SPI AVIP Testbench Architecture has divided into the two top modules as HVL and HDL top as shown in below fig 2

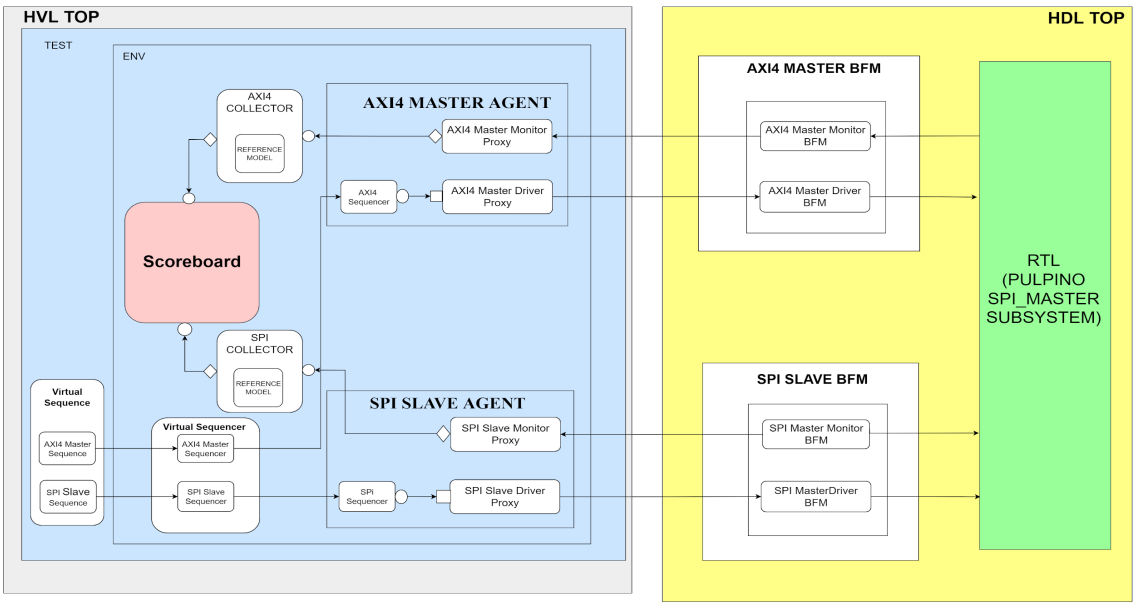


Fig 2.1 pulpino_spi_master_avip Architecture

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 AXI4 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 pulpino__spi_master__ip_verification.git clone of the repository in the terminal that is pulpino__spi_master__subsystem_verification

   gitclone

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

   cd pulpino__spi_master__subsystem_verification.git
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 pulpino__spi_master__subsystem_verification repository

   git fetch
7. Check all branches present in the pulpino__spi_master__subsystem_verification 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, pulpino__spi_master__subsystem_verification
[pulpino__spi_master__subsystem_verification 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 pulpino__spi_master__subsystem_verification/sim/questasim

   Note: To Compile, simulate, regression and for coverage you must be in the specified path i.e, *pulpino__spi_master__subsystem_verification/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 compile.log to view the compiled files
   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: *basic_write_read_reg_test/basic_write_read_reg_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 basic_write_read_reg_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

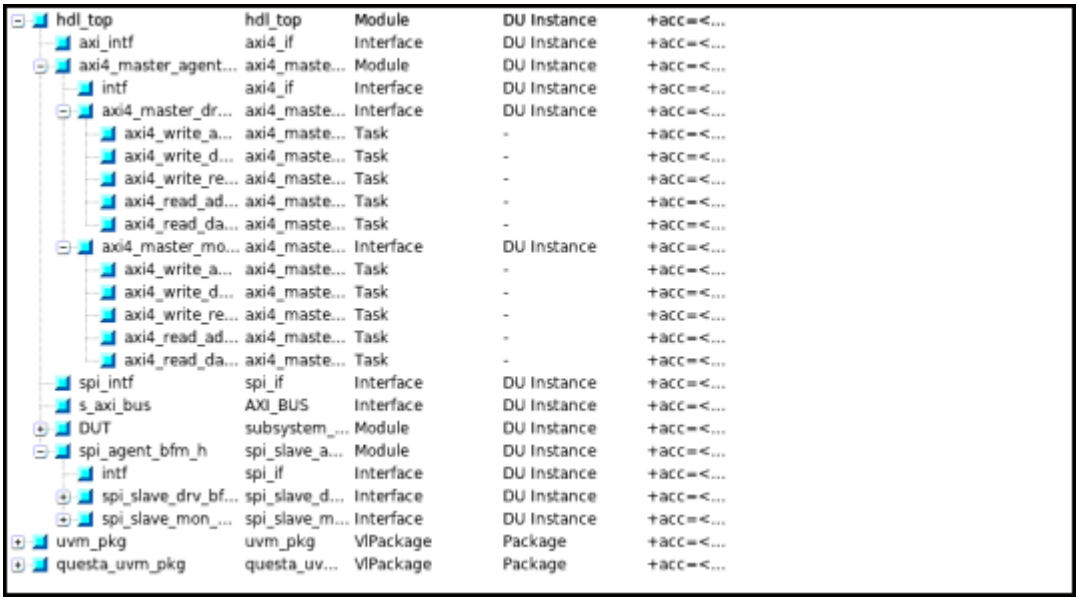


Fig 3.2 Questasim WLF window

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

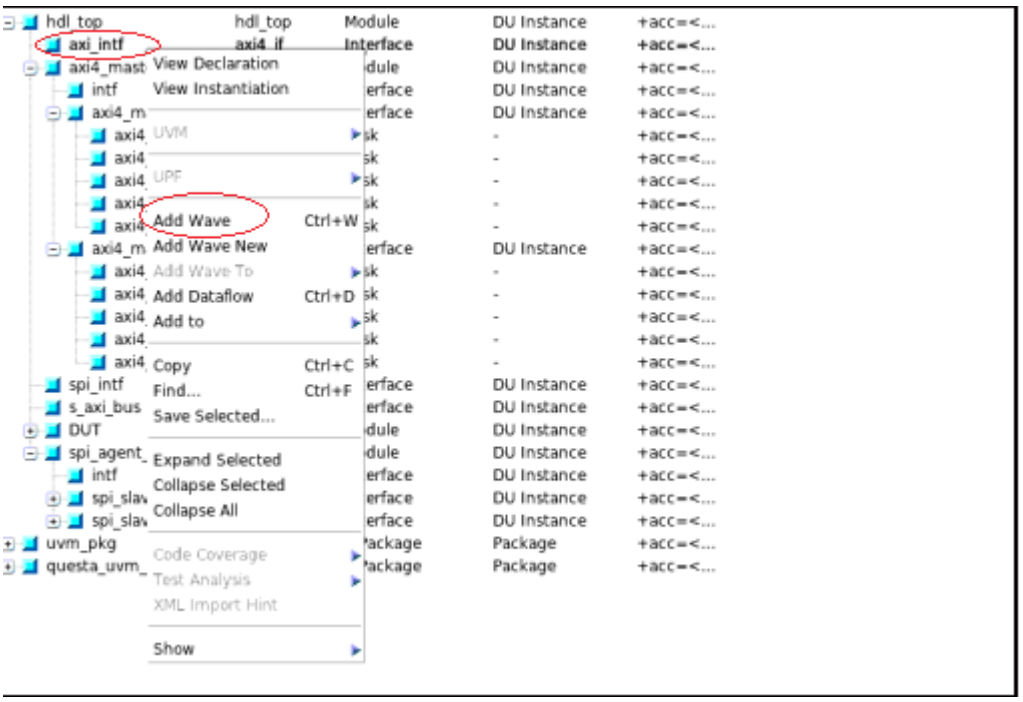


Fig 3.2 Screenshot of adding waves in wave window

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

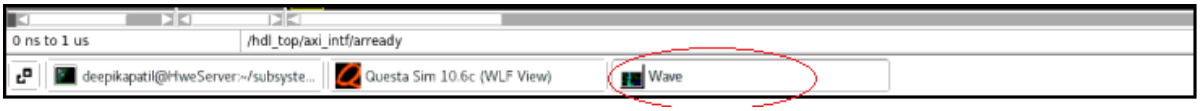


Fig 3.3 Wave window

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

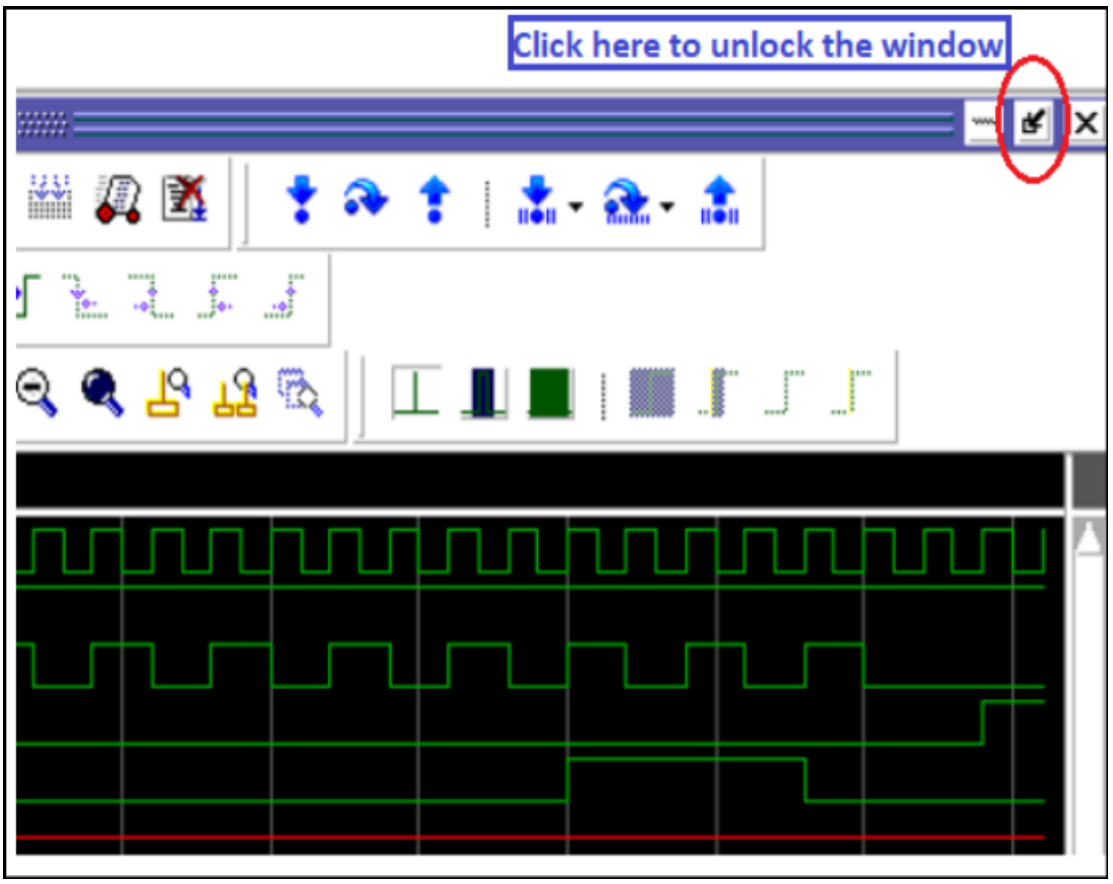


Fig 3.4 Screenshot of unlocking the wave window

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

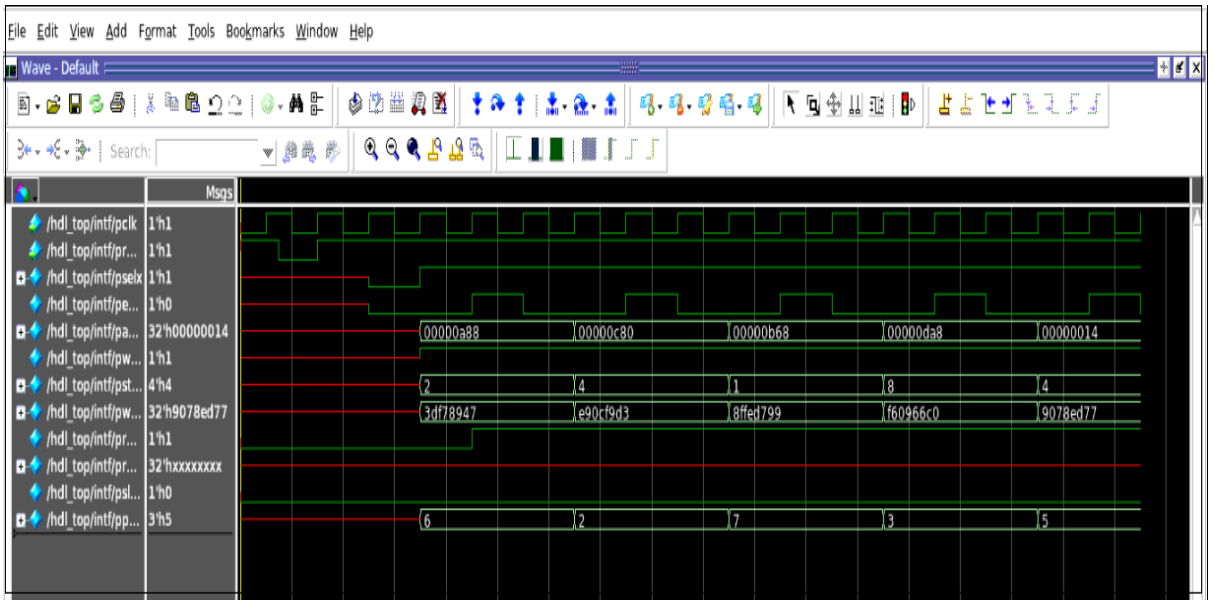


Fig 3.5 Screenshot of unlocked wave window with signals

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

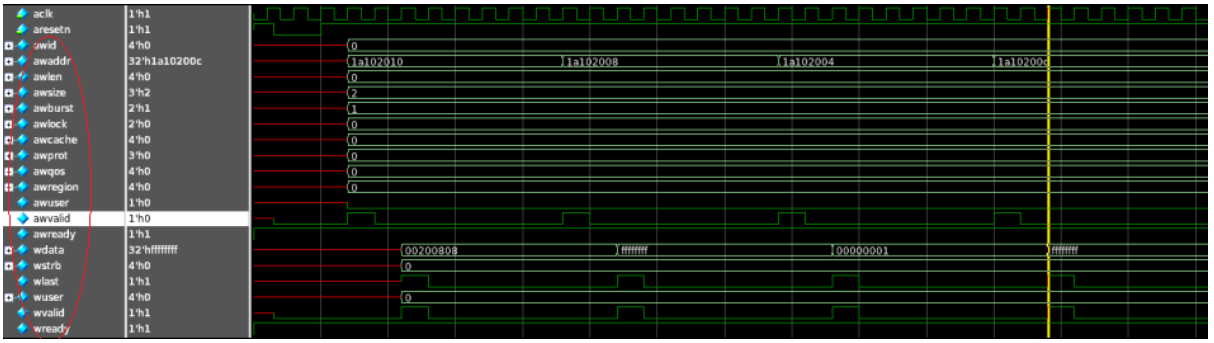


Fig 3.6 Screenshot showing way to see the name of signals

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=regression.list*

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

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

   ls

```
axi4_simple_reg_test_22032022-182024
basic_write_read_reg_test_22032022-182029
basic_write_reg_test_22032022-182035
```

Fig 3.7 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 *basic_write_read_reg_test*

   Go inside the directory of interested testcase with the date

   basic_write_reg_test_22032022-182035

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

   basic_write_read_reg_test.log

Path:

basic_write_read_reg_test_22032022-182035 /basic_write_read_reg_test.log

3.2.4 Coverage

1. To see coverage

a. After simulating

For the individual test, use the command firefox

firefox basic_write_read_reg_test/html_cov_report/index.html &

Ex: *firefox basic_write_read_reg_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 *basic_write_read_reg_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

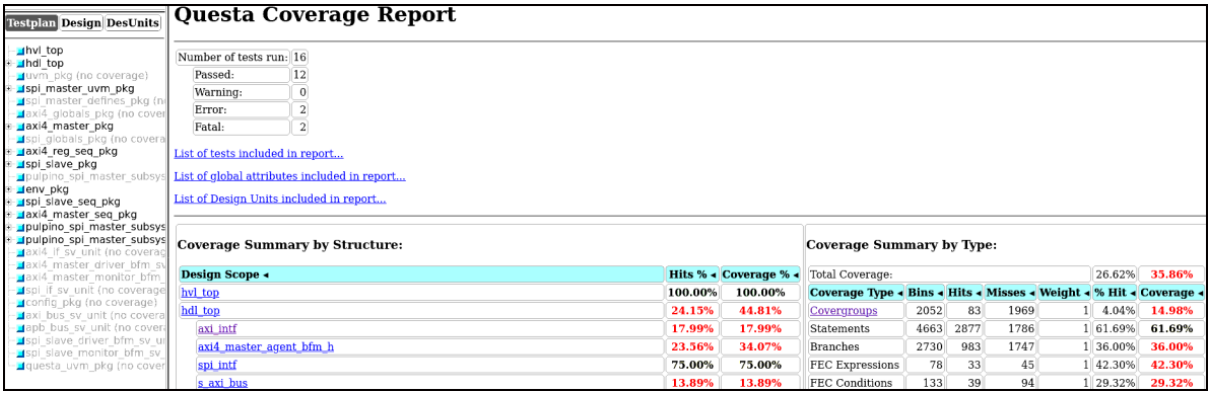
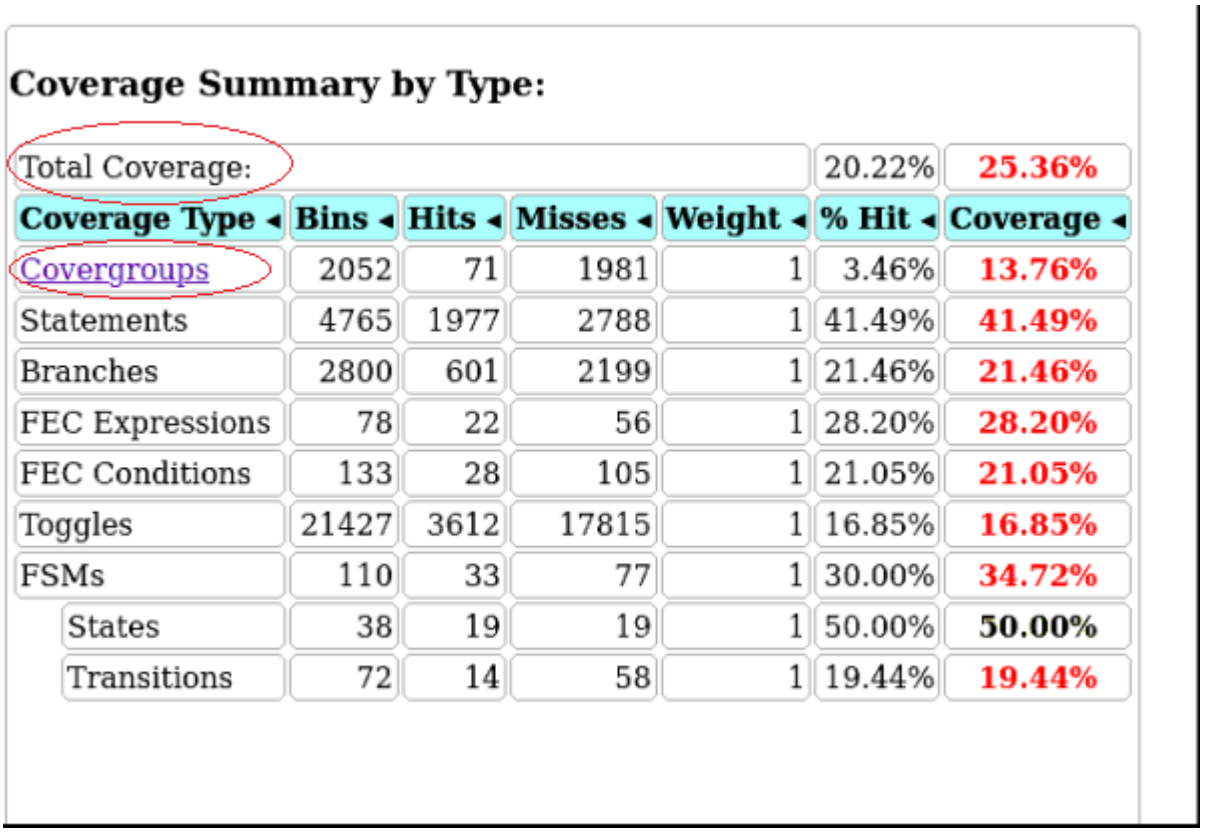


Fig 3.8 Coverage Report

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

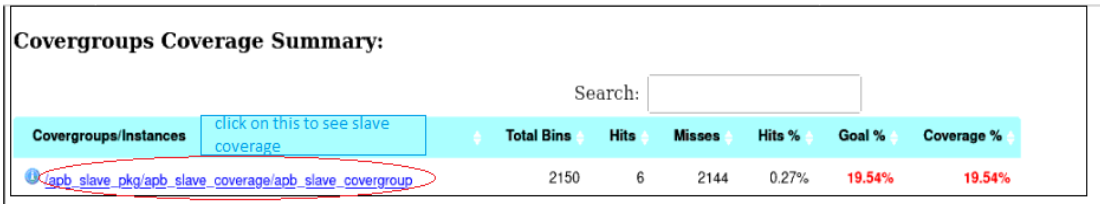


Coverage Summary by Type:

Total Coverage:					20.22%	25.36%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Covergroups	2052	71	1981	1	3.46%	13.76%
Statements	4765	1977	2788	1	41.49%	41.49%
Branches	2800	601	2199	1	21.46%	21.46%
FEC Expressions	78	22	56	1	28.20%	28.20%
FEC Conditions	133	28	105	1	21.05%	21.05%
Toggles	21427	3612	17815	1	16.85%	16.85%
FSMs	110	33	77	1	30.00%	34.72%
States	38	19	19	1	50.00%	50.00%
Transitions	72	14	58	1	19.44%	19.44%

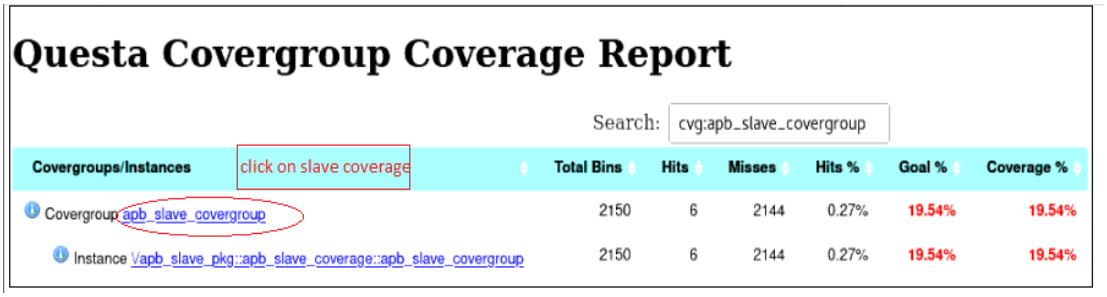
3.9 Screenshot of opening covergroups

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



3.10 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



3.11 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

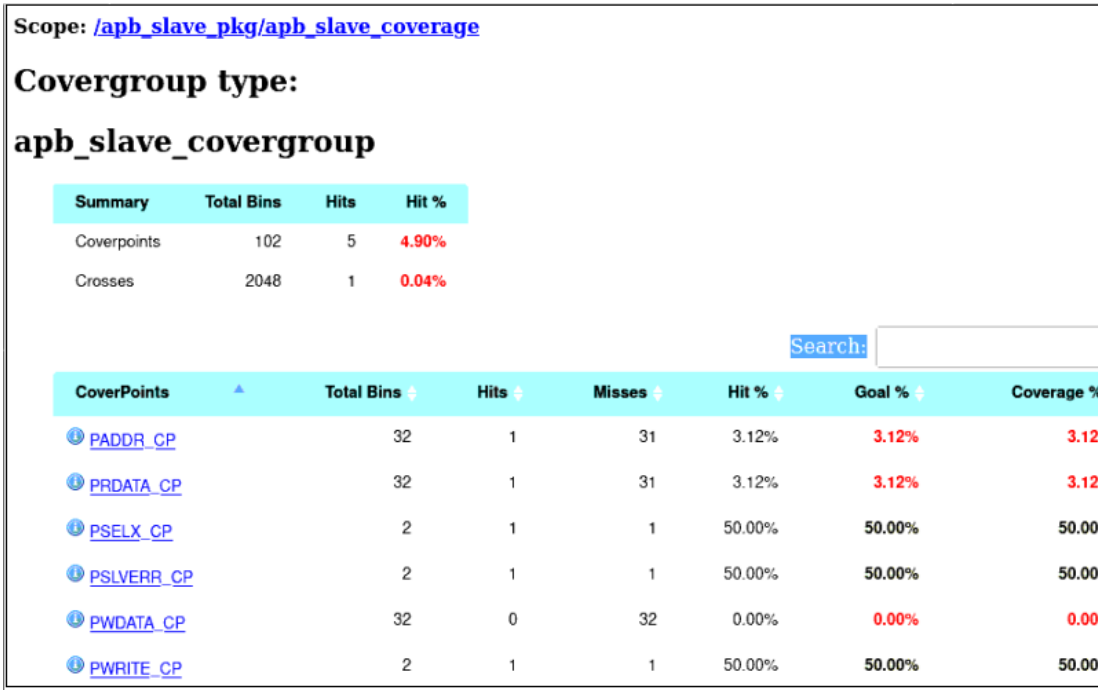
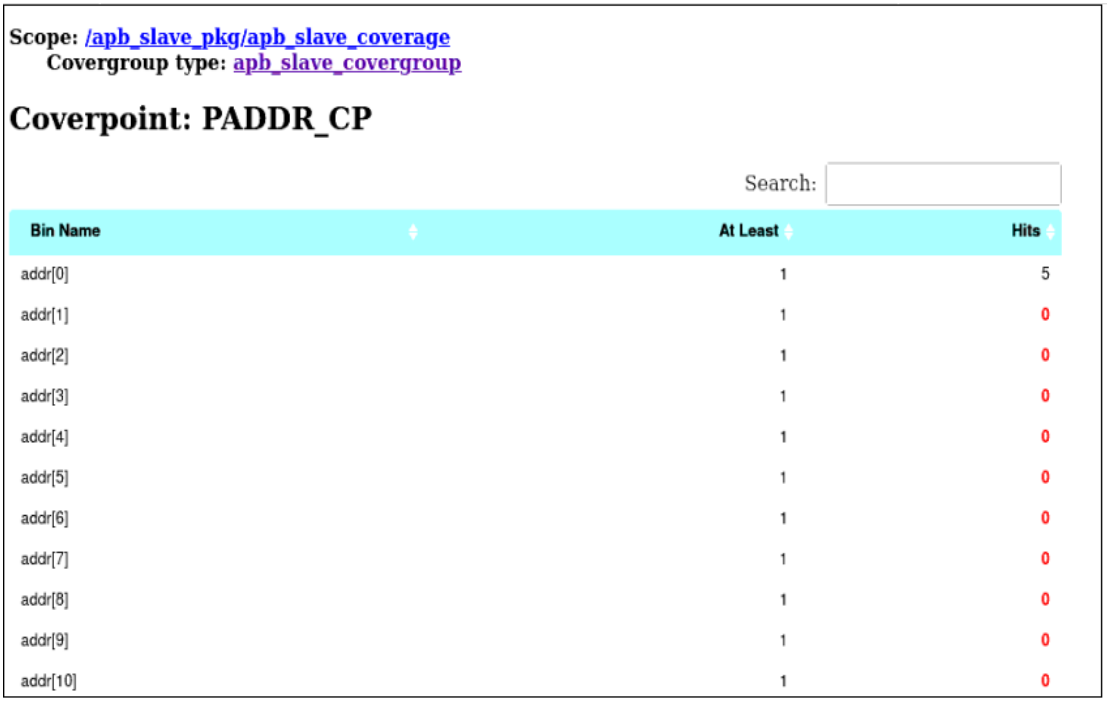


Fig 3.12 Screenshot of Coverpints and cross coverpoints

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. 3.13 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, *pulpino__spi_master__subsystem_verification/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 3.14 Usage of make command in cadence

3.3.1 Compilation

1. Use the following command to compile

   make compile

2. Open the log file compile.log to view the compiled files
   vim 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
   pulpino_spi_master_subsystem_verification_compile/src/hvl_top/testlists/regression.list

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

   Ex: *basic_write_read_reg_test/basic_write_read_reg_test.log*

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

```
--------------------------------------------------------------
------------------- Simulation Report --------------------------

UVM Fatal
Number of demoted UVM_FATAL reports  :    0
Number of caught UVM_FATAL reports   :    0
UVM_FATAL :    0

UVM Errors
Number of demoted UVM_ERROR reports  :    0
Number of caught UVM_ERROR reports   :    0
UVM_ERROR :    0

UVM Warnings
Number of demoted UVM_WARNING reports:    0
Number of caught UVM_WARNING reports :    0
UVM_WARNING :    0

Testname: spi_simple_fd_8b_test
Log file path: spi_simple_fd_8b_test/spi_simple_fd_8b_test.log
Waveform: vsim -view spi_simple_fd_8b_test/waveform.wlf &
```

Fig 3.15 Simulation report in cadence

3. To view waveform

   simvision waves.shm/

1. After running the simvision command , we will be able to see the waveform window as shown in the figure 3.16

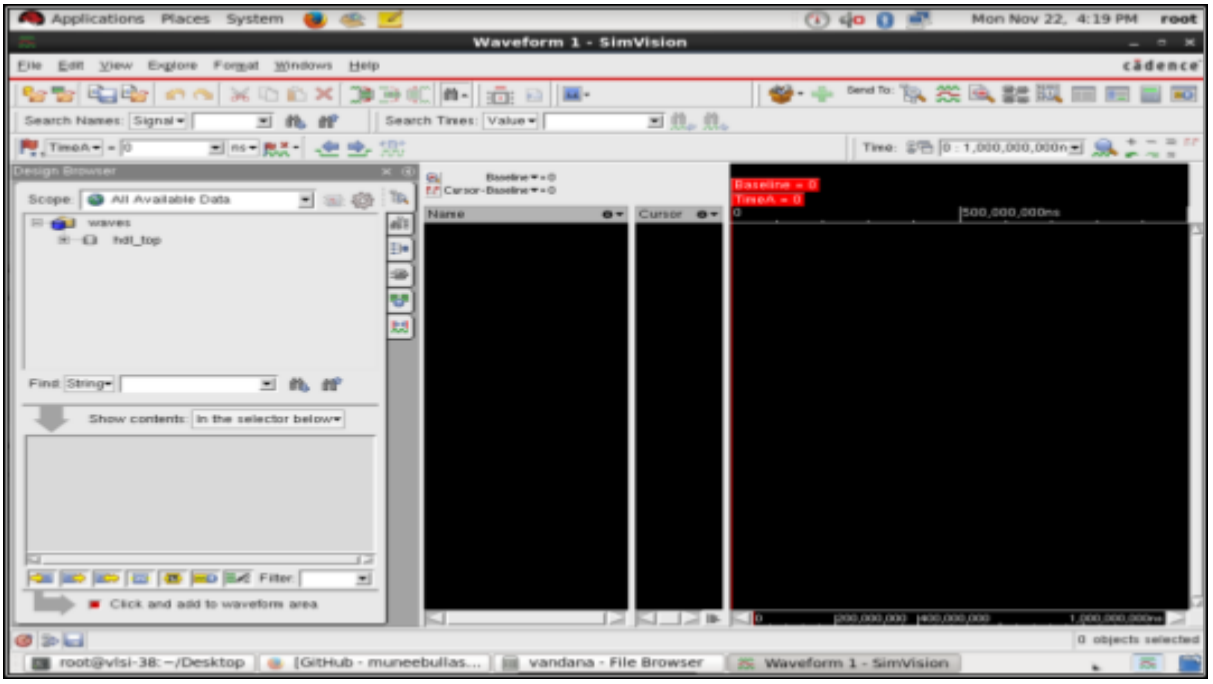


Fig 3.16 Cadence waveform window

2. From the right side of the waveform window, right click on to the intf where all signals are present and select send to waveform window as shown in fig 3.17

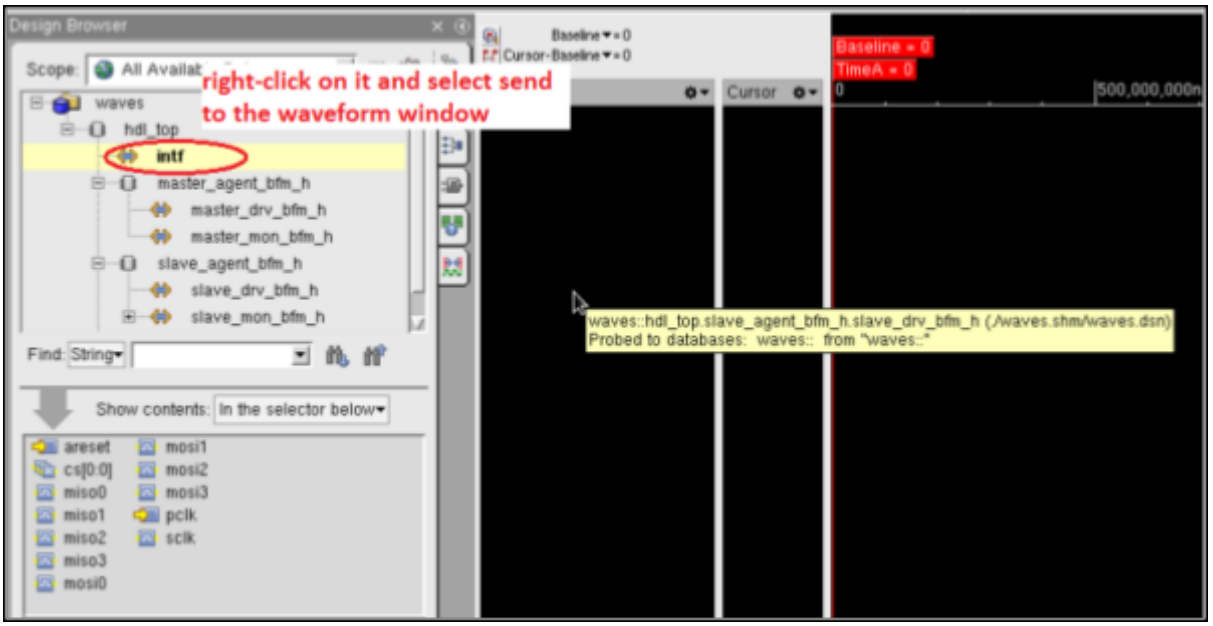


Fig 3.17

3. All signals appears in the waveform window as shown in the fig 3.18

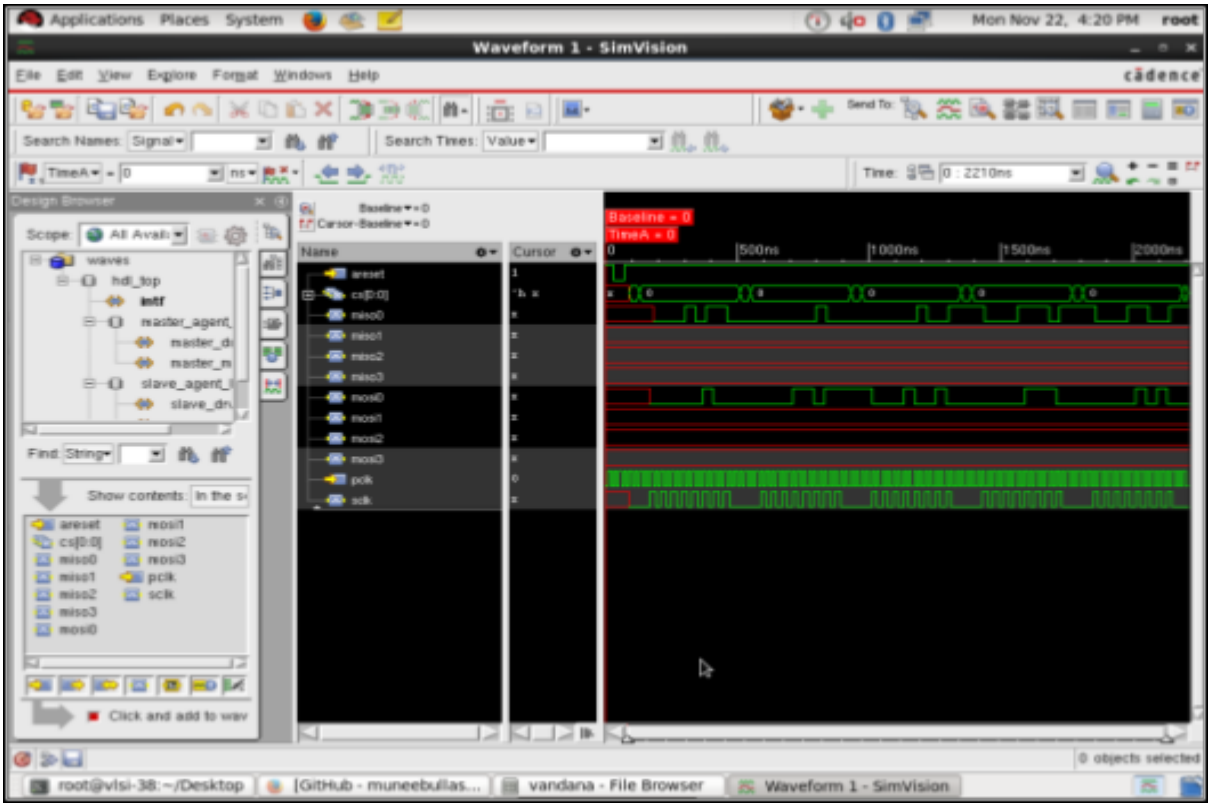


Fig 3.18

3.3.3 Coverage

1. To view coverage, type the imc command as below
   imc -load cov_work/scope/test/ &
2. After running the imc command the window appears as shown in the fig 3.19

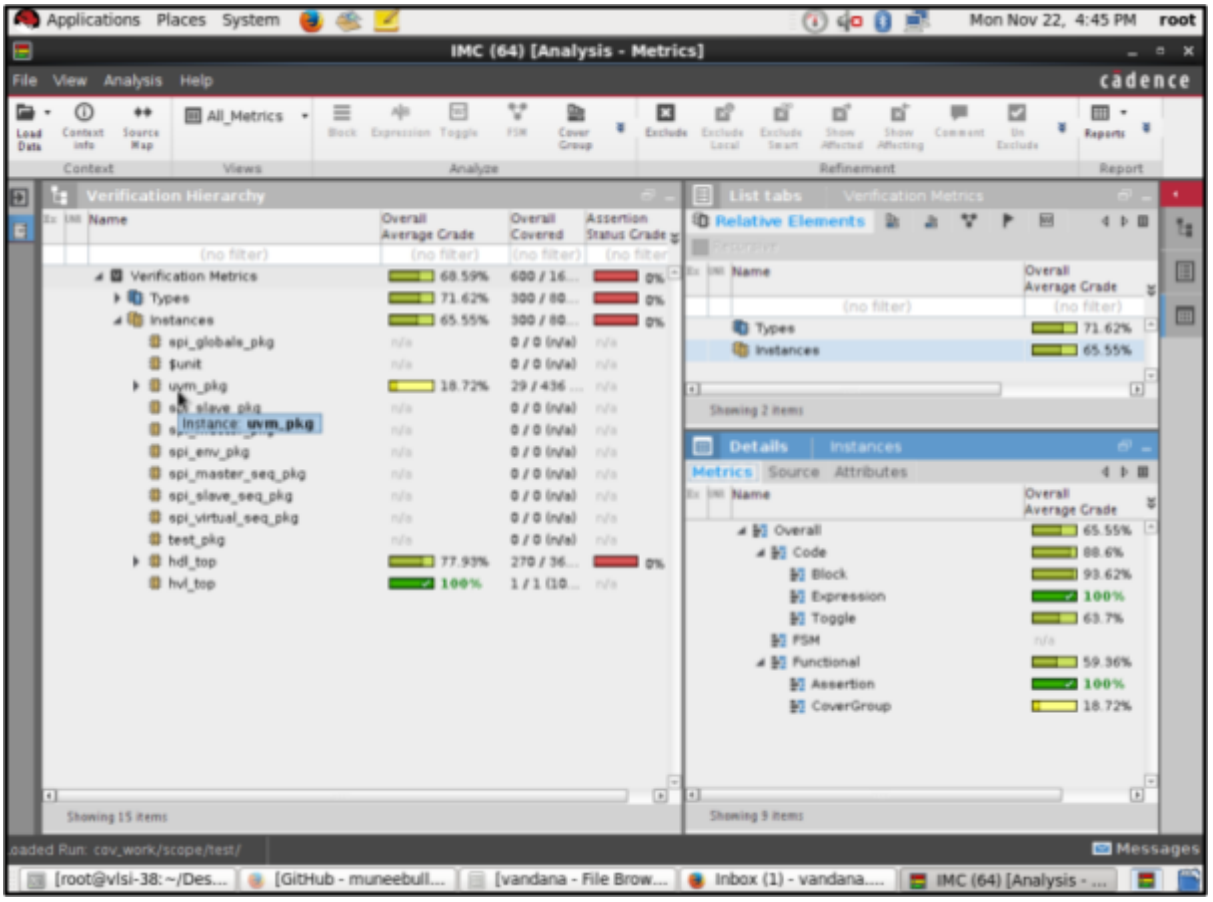


Fig 3.19

3. Click on the black arrow mark pointing to the uvm pkg as shown in the fig 3.20 later then click on arrow mark pointing to env_h , then on master_agent_h then right click on master_cov_h and select

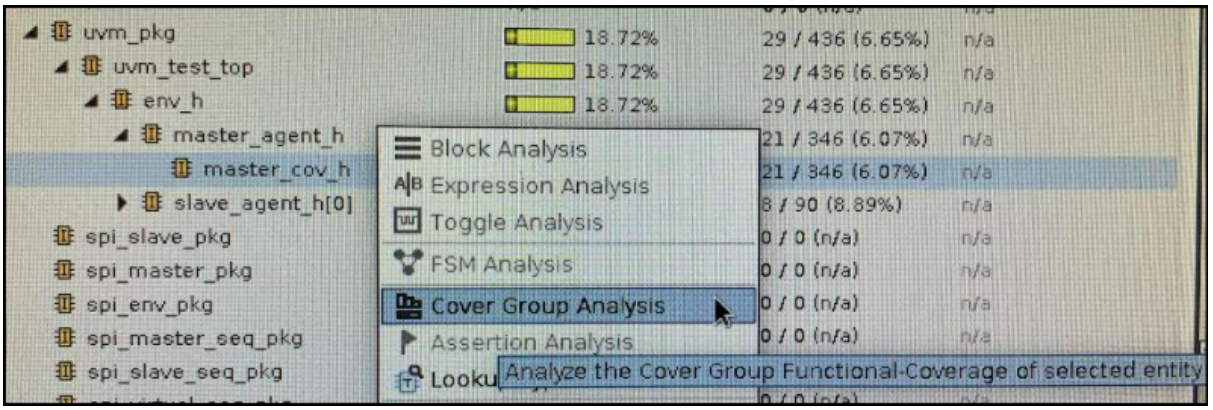
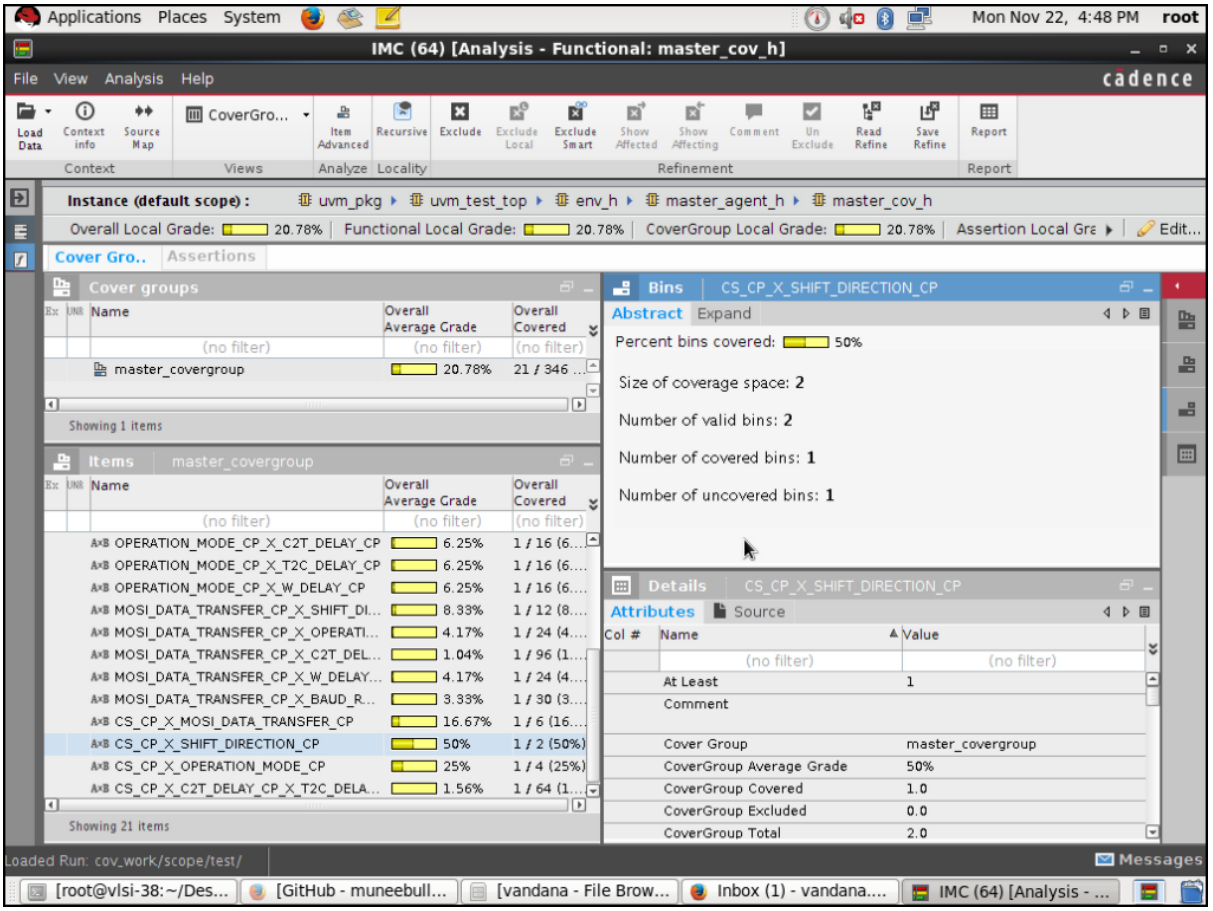
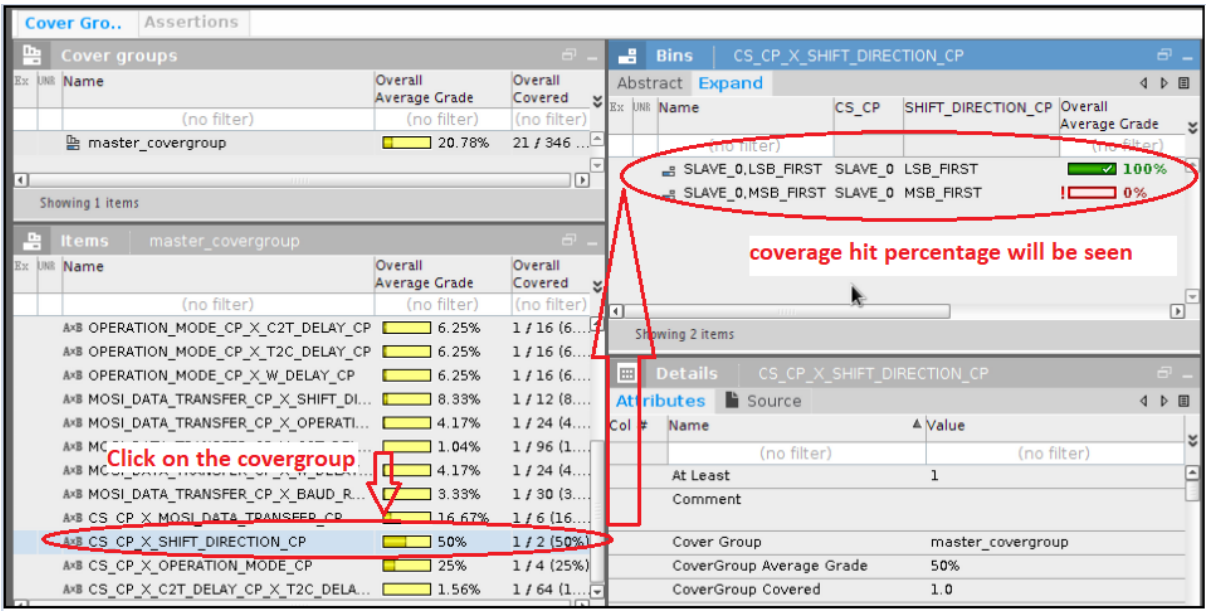


Fig 3.20

4. Covergroup analysis window appears as shown in the fig.3.21



5. Click on any of the covergroup to see the coverage percentage of that covergroup as shown in the fig 3.22



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 basic_write_read_reg_test has been used as an example test case in order to show the below debugging flow of the AXI4 protocol and all the info's have been runned using **UVM_HIGH** verbosity

4.1 AXI4 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.

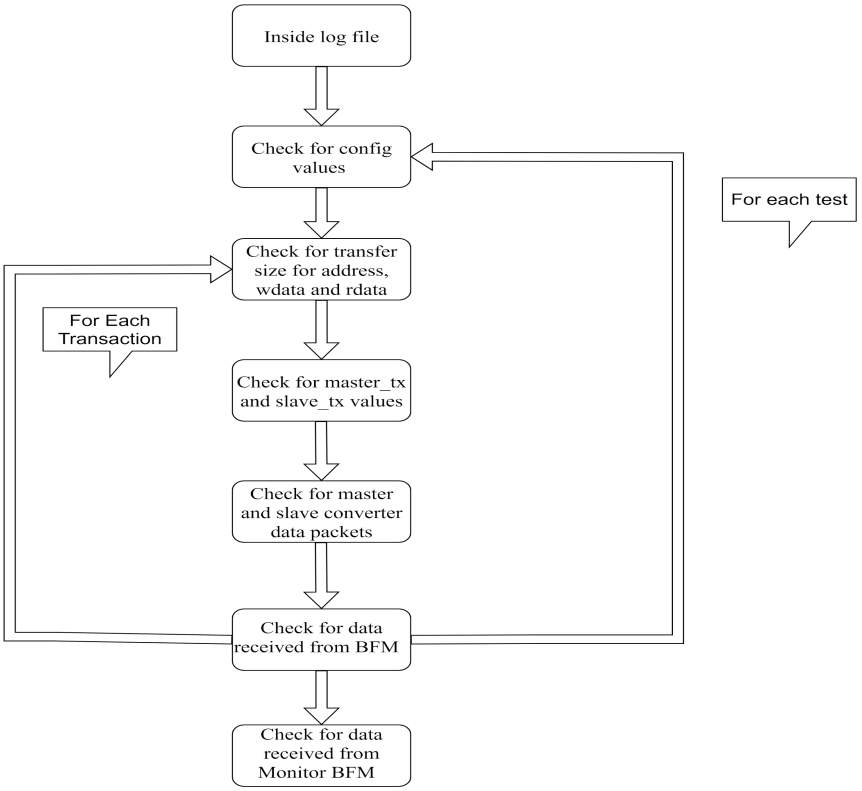


Fig 4.1 Debugging flow

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

```
UVM_INFO ../../src/dv/hvl_top/test//base_test.sv(80) @ 0: uvm_test_top [basic_write_read_reg_test
axi4_master_agent_CONFIG
---------------------------------------------------------------------------
Name                                Type                      Size  Value
---------------------------------------------------------------------------
axi4_master_agent_config            axi4_master_agent_config  -     @497
  is_active                         string                    10    UVM_ACTIVE
  has_coverage                      integral                  1     1
  wait_count_write_address_channel  integral                  32    'd0
  wait_count_write_data_channel     integral                  32    'd0
  wait_count_read_address_channel   integral                  32    'd0
  outstanding_write_tx              integral                  32    'd0
  outstanding_read_tx               integral                  32    'd0
```

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
spi_mode	which is of enum type and it will indicate which type of configurations related to cpol and cpha the test running
shift_dir	which is of enum type and it tells about whether an MSB bit should start the transfer or an LSB bit should begin the transfer.
has_coverage	Which indicates the coverage connection
Slave_id	Tells which slave is selected

```
pulpino_spi_master_subsystem_SLAVE_CONFIG[0]
------------------------------------------------------------------
Name                    Type                    Size  Value
------------------------------------------------------------------
slave_agent_config[0]   spi_slave_agent_config  -     @502
  is_active             string                  10    UVM_ACTIVE
  slave_id              integral                2     'd0
  spi_mode              string                  11    CPOL0_CPHA1
  shift_dir             string                  9     MSB_FIRST
  has_coverage          integral                1     1
  spi_type              string                  10    SIMPLE_SPI
```

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

```
env_CONFIG
-----------------------------------------
Name                 Type        Size  Value
-----------------------------------------
env_cfg_h            env_config  -     @496
  has_scoreboard     integral    1     1
  has_virtual_seqr   integral    1     1
  no_of_spi_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

AXI4 follows one specific rule related to the transfer size of awdata and ardata.

The number of bits or bytes transferred by the awaddr to the write data bus size is one byte.

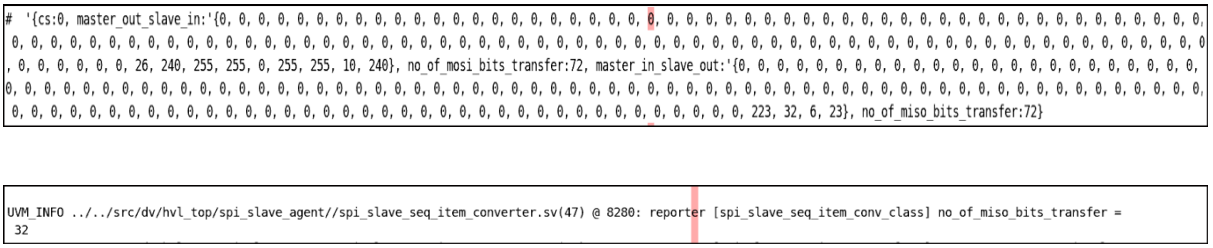


Fig 4.5 Transfer size for mosi and miso

Figure 4.5 shows the transfer size of Awdata or wdata.

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 become high the data will sampled on the same clock edge.

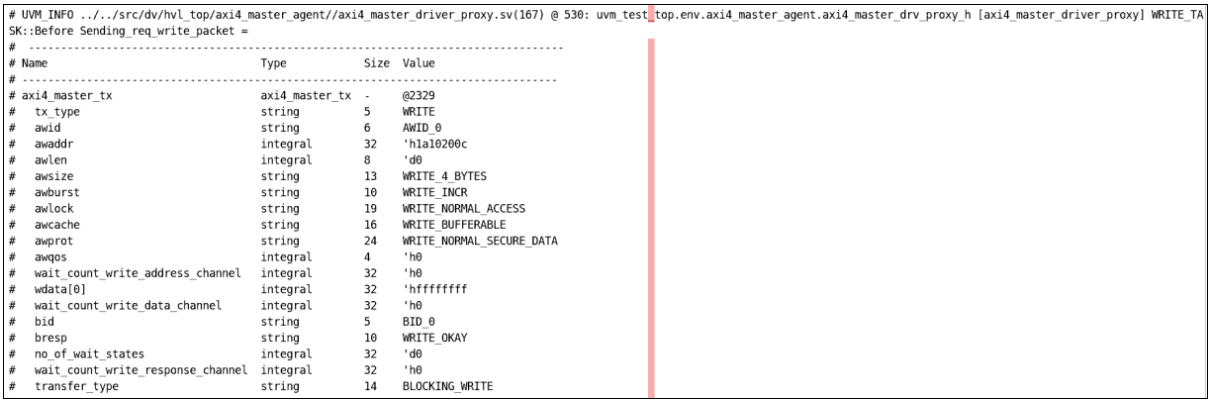


Fig 4.6 master_tx values

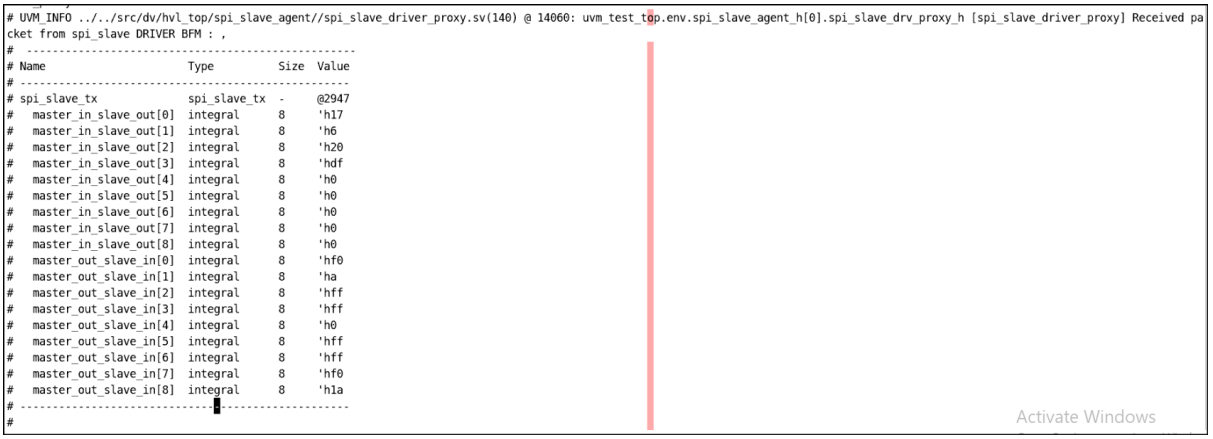


Fig 4.7 slave_tx values

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.

Fig 4.8 converted data of master req

```
# UVM_INFO ../../src/dv/hvl_top/spi_slave_agent//spi_slave_driver_proxy.sv(129) @ 1330: uvm_test_top.env.spi_slave_agent_h[0].spi_slave_drv_proxy_h [spi_slave_driver_proxy] Received pac
ket from spi_slave sequencer : ,
#  -----------------------------------------------------------------------------------------------------------------------------
# Name                         Type         Size  Value
# -----------------------------------------------------------------------------------------------------------------------------
# req                          spi_slave_tx  -     @2550
#   begin_time                 time         64    1330
#   depth                      int          32    'd2
#   parent sequence (name)     string       24    spi_fd_basic_slave_seq_h
#   parent sequence (full name) string      79    uvm_test_top.env.spi_slave_agent_h[0].spi_slave_seqr_h.spi_fd_basic_slave_seq_h
#   sequencer                  string       54    uvm_test_top.env.spi_slave_agent_h[0].spi_slave_seqr_h
#   master_in_slave_out[0]     integral     8     'hbc
#   master_in_slave_out[1]     integral     8     'h6b
#   master_in_slave_out[2]     integral     8     'h46
#   master_in_slave_out[3]     integral     8     'h80
```

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 awaddr, awvalid, awready, awdtata signals and samples the ardata, arvalid, arready depending on configurations of master and similarly slave driver BFM will drive the ardata, arvalid, arready signal and samples the awaddr, awvalid, awready ,awdata 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/dv/hdl_top/axi4_master_agent_bfm/axi4_master_driver_bfm.sv(117) @ 7550: reporter [AXI4_MASTER_DRIVER_BFM] data_write_packet=
# '{awid:0, awaddr:437264396, awlen:0, awsize:2, awburst:1, awlock:0, awcache:0, awqos:0, awregion:0, awuser:0, awprot:0, awvalid:0, awready:0, wdata:'{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40272
52735}, wstrb:'{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, wuser:0, wlast:0, bid:0, bresp:0, buser:0, bvalid:0, tx_type:0, wait_count_write_address_channel:0, wait_count_write_data_channel:0, wait
_count_write_response_channel:0, outstanding_write_tx:0, no_of_wait_states:0}
```

Fig 4.10 master bfm_struct data

```
# UVM_INFO ../../src/dv/hvl_top/spi_slave_agent//spi_slave_driver_proxy.sv(168) @ 7060: uvm_test_top.env.spi_slave_agent_h[0].spi_slave_drv_proxy_h [spi_slave_driver_proxy] BFM STRUCT D
ATA : ,
#  '{cs:0, master_out_slave_in:'{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 26, 240, 255, 255, 0, 255, 255, 10, 240}, no_of_mosi_bits_transfer:72, master_in_slave_out:'{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 70, 107, 188}, no_of_miso_bits_transfer:72}
```

Fig 4.11 slave bfm_struct data

The fig 4.10 and 4.11 shows the data with respect to mosi and miso signals of both master and slave end before converting back to req using to_class converter.

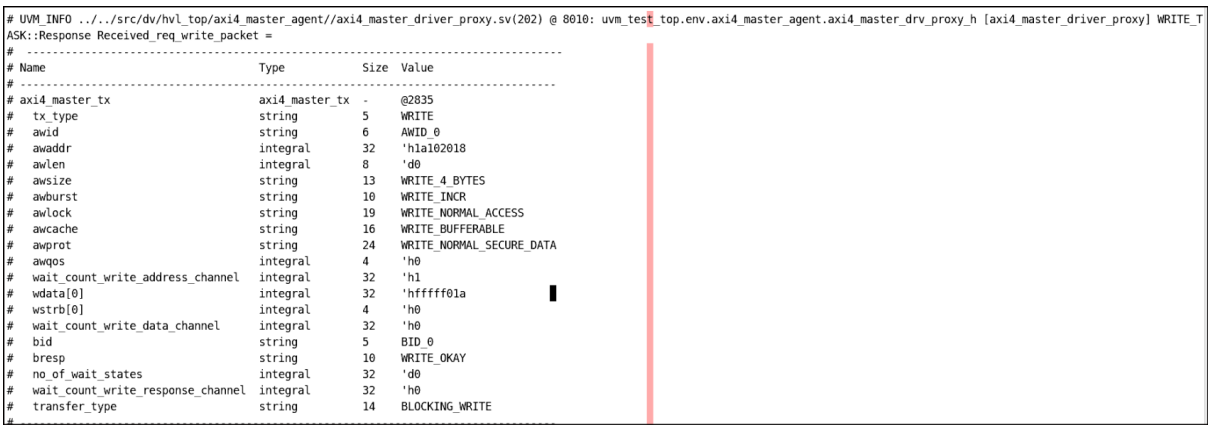


Fig 4.12 master_driver_bfm values

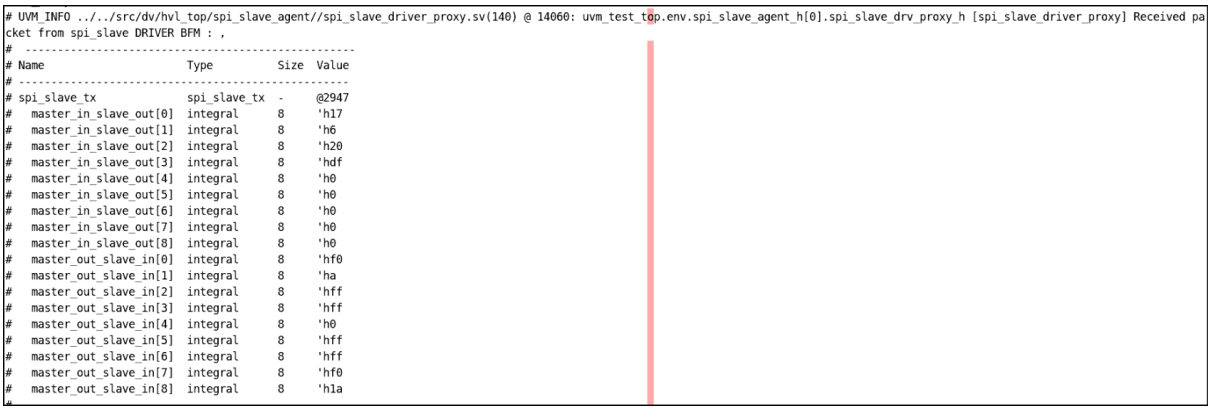


Fig 4.13 slave_driver_bfm values

Fig 4.12 and 4.13 shows the mosi and miso 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

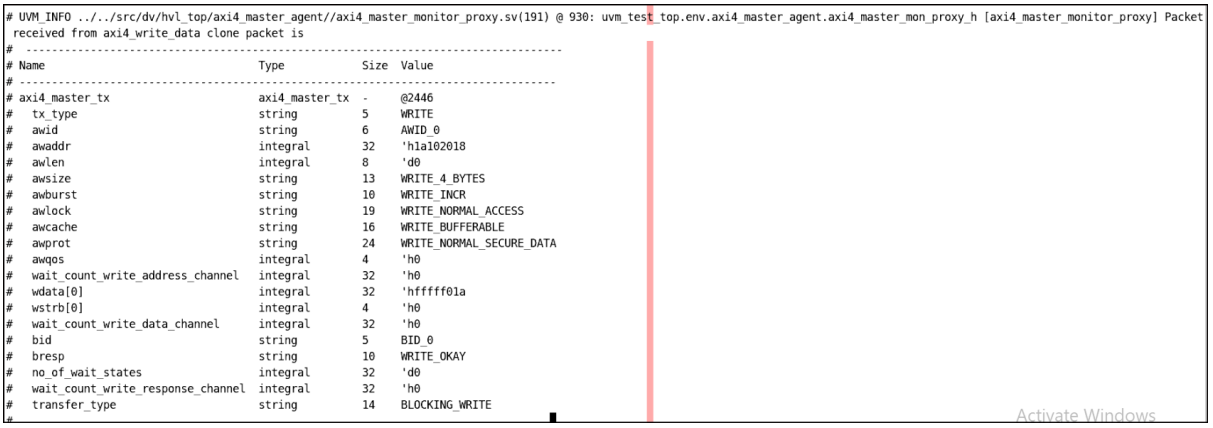


Fig 4.14 master_monitor values

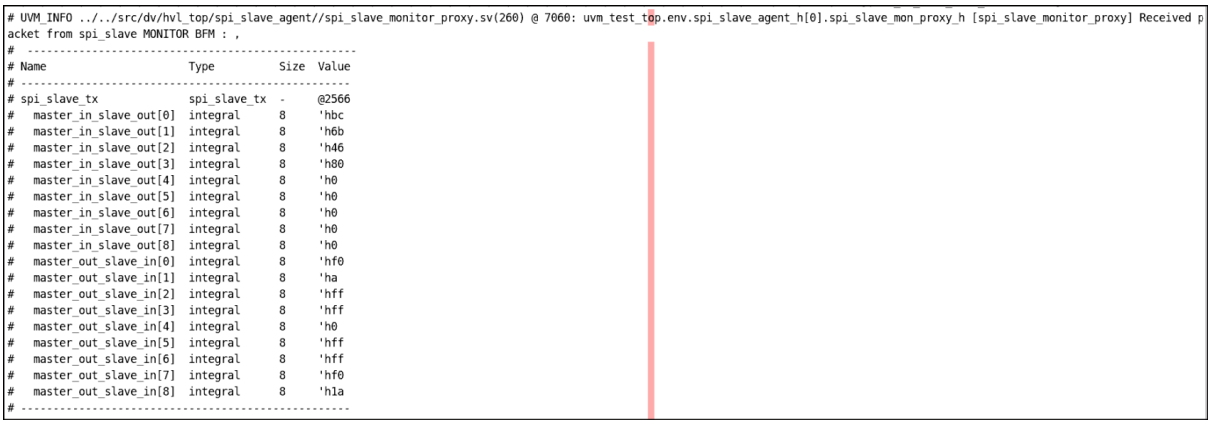


Fig 4.15 slave_monitor values

4.2 Scoreboard Checks

And finally we have scoreboard checks which basically compares the awaddr,data_width data of master with the slave side

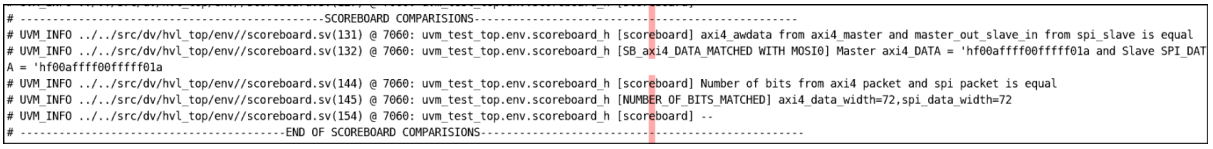


Fig 4.16 scoreboard_checks

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/dv/hvl_top/apb_master_agent//apb_master_coverage.sv(151) @ 10030: uvm_test_top.pulpino_spi_master_ip_env.apb
_master_agent.apb_master_cov_h [apb_master_coverage] APB Master Agent Coverage = 29.34 %
```

Fig 4.17 coverage for master

```
  UVM_INFO ../../src/dv/hvl_top/spi_slave_agent//spi_slave_coverage.sv(242) @ 10030: uvm_test_top.pulpino_spi_master_ip_env.spi_s
lave_agent_h[0].spi_slave_cov_h [spi_slave_coverage] spi_slave Agent Coverage = 9.38 %
```

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 &

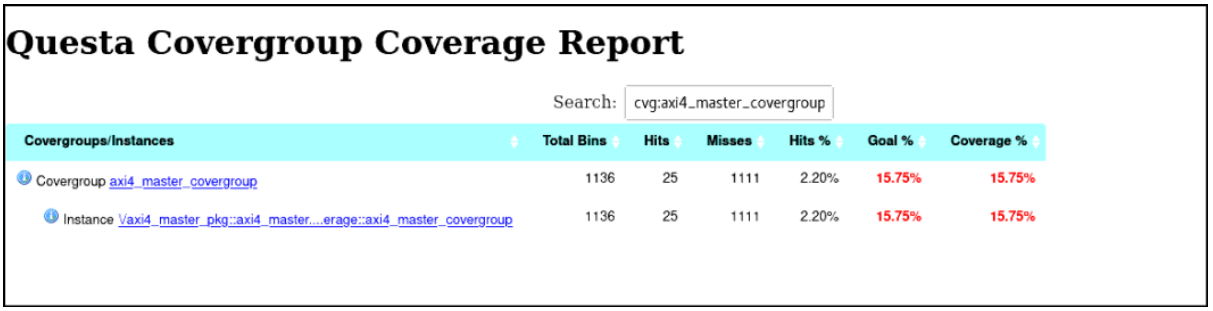
Inside that check for covergroups in the coverage summary then check for the instance created for master and slave coverage

Covergroups Coverage Summary:

Search:

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
/spi_master_uvm_pkg/spi_master_apb_if__status/cg_vals	26	6	20	23.07%	42.70%	42.70%
/spi_master_uvm_pkg/spi_master_apb_if__clkdiv/cg_vals	64	1	63	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__spicmd/cg_vals	64	1	63	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__spiadr/cg_vals	64	1	63	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__spilen/cg_vals	192	3	189	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__spidum/cg_vals	128	2	126	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__txfifo/cg_vals	64	1	63	1.56%	1.56%	1.56%
/spi_master_uvm_pkg/spi_master_apb_if__rxfifo/cg_vals	64	0	64	0.00%	0.00%	0.00%
/spi_master_uvm_pkg/spi_master_apb_if__intcfg/cg_vals	132	6	126	4.54%	18.75%	18.75%
/spi_master_uvm_pkg/spi_master_apb_if_coverage/ra_cov	28	17	11	60.71%	61.98%	61.98%
/spi_slave_pkg/spi_slave_coverage/spi_slave_covergroup	90	2	88	2.22%	9.37%	9.37%
/apb_master_pkg/apb_master_coverage/apb_master_covergroup	145	14	131	9.65%	29.34%	29.34%

Fig 4.19 master and slave coverage



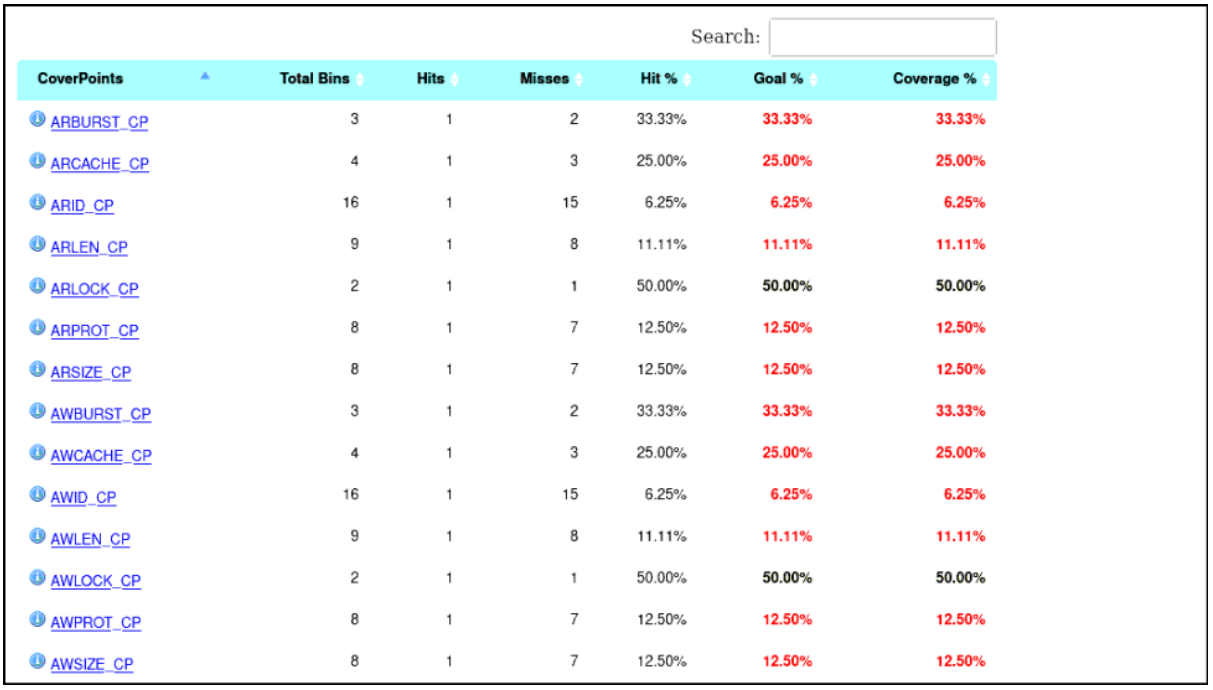
Questa Covergroup Coverage Report

Search: cvg:axi4_master_covergroup

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
Covergroup axi4_master_covergroup	1136	25	1111	2.20%	15.75%	15.75%
Instance \axi4_master_pkg::axi4_master....erage::axi4_master_covergroup	1136	25	1111	2.20%	15.75%	15.75%

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 configuration modes, cs, delays, data widths,and baud rate.



Search:

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
ARBURST_CP	3	1	2	33.33%	33.33%	33.33%
ARCACHE_CP	4	1	3	25.00%	25.00%	25.00%
ARID_CP	16	1	15	6.25%	6.25%	6.25%
ARLEN_CP	9	1	8	11.11%	11.11%	11.11%
ARLOCK_CP	2	1	1	50.00%	50.00%	50.00%
ARPROT_CP	8	1	7	12.50%	12.50%	12.50%
ARSIZE_CP	8	1	7	12.50%	12.50%	12.50%
AWBURST_CP	3	1	2	33.33%	33.33%	33.33%
AWCACHE_CP	4	1	3	25.00%	25.00%	25.00%
AWID_CP	16	1	15	6.25%	6.25%	6.25%
AWLEN_CP	9	1	8	11.11%	11.11%	11.11%
AWLOCK_CP	2	1	1	50.00%	50.00%	50.00%
AWPROT_CP	8	1	7	12.50%	12.50%	12.50%
AWSIZE_CP	8	1	7	12.50%	12.50%	12.50%

Fig 4.21 master_coverage coverpoint

Figure 4.21 shows all the coverpoints included in master coverage

Search:

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
ARBURST_CP_X_ARLEN_CP_X_ARSIZE_CP	216	1	215	0.46%	0.46%	0.46%
ARLENGTH_CP_X_ARSIZE_X_ARBURST	216	1	215	0.46%	0.46%	0.46%
AWBURST_CP_X_AWLEN_CP_X_AWSIZE_CP	216	1	215	0.46%	0.46%	0.46%
AWLENGTH_CP_X_AWSIZE_X_AWBURST	216	1	215	0.46%	0.46%	0.46%

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 configuration modes and shift directions and different data widths

CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
MISO_DATA_TRANSFER_CP	6	0	6	0.00%	0.00%	0.00%
MOSI_DATA_TRANSFER_CP	6	0	6	0.00%	0.00%	0.00%
OPERATION_MODE_CP	4	1	3	25.00%	25.00%	25.00%
SHIFT_DIRECTION_CP	2	1	1	50.00%	50.00%	50.00%

Fig 4.23 salve_coverage coverpoint

Figure 4.23 shows all the coverpoints included in slave coverage

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
OPERATION_MODE_X_MASTER_IN_slave_OUT	24	0	24	0.00%	0.00%	0.00%
OPERATION_MODE_X_MASTER_OUT_slave_IN	24	0	24	0.00%	0.00%	0.00%
SHIFT_DIRECTION_X_MASTER_IN_slave_OUT	12	0	12	0.00%	0.00%	0.00%
SHIFT_DIRECTION_X_MASTER_OUT_slave_IN	12	0	12	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

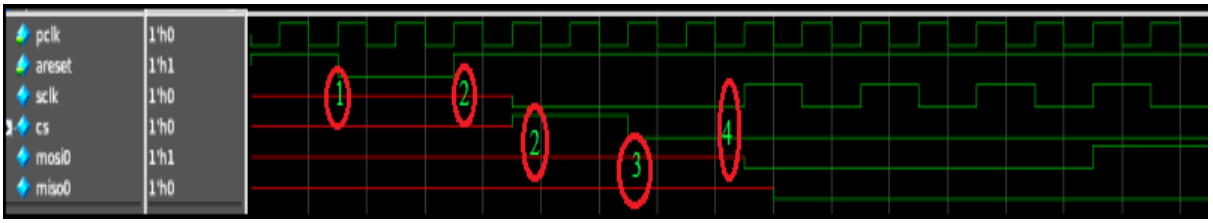


Fig 4.30 wave form for 32 bit full duplex test with initial conditions

Figure 4.30 shows the waveform for 32 bit full duplex mode with all the required signals such as mosi0, miso0, sclk, chip select(cs), and reset.

1. In the waveform, initially check for the generation of the system clock(pclk), after every 10ns it will be toggled. Once the pclk is done check for the reset condition(Active low reset) if the reset is low the other signals such as sclk, mosi, miso and chip select should be in an unknown state.

2. Once the reset is high at the next posedge of pclk the cs and sclk should be in idle state.

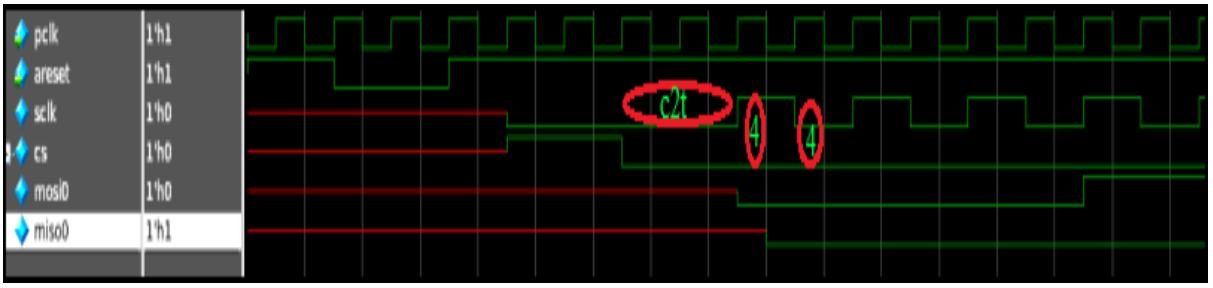


Fig 4.31 wave form for 32 bit full duplex test with c2t delay

3. After the idle state is completed, chip select will go low and sclk still remains as cpol. Since c2t delay is 1 and baud rate is 2 the sclk will generate after 2 clock cycles of pclk.

4. Once sclk generates, at every posedge of sclk mosi data will be driven onto the mosi signal which sends data to the slave mosi. And miso will be sampled at the negedge of the sclk.

Fig 4.32 wave form for 32 bit full duplex test with all the delays

5. After completion of driving and sampling all the mosi and miso bits there should be a delay of 2 pclks between the sclk and next chip select(t2c delay = 1). Once the transaction completes chip select goes high for 2 clk cycles of pclk(wdelay = 1) then the next transaction will start.

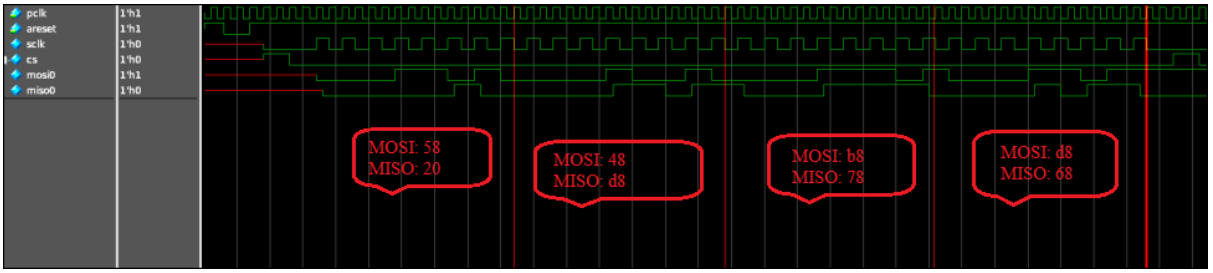


Fig 4.33 wave form for 32bit data transfer(1st transaction)

Fig 4.33 shows the waveform for whole 32 bit full duplex transfer with c2t delay and cpol_0 and cpha_0 configurations.

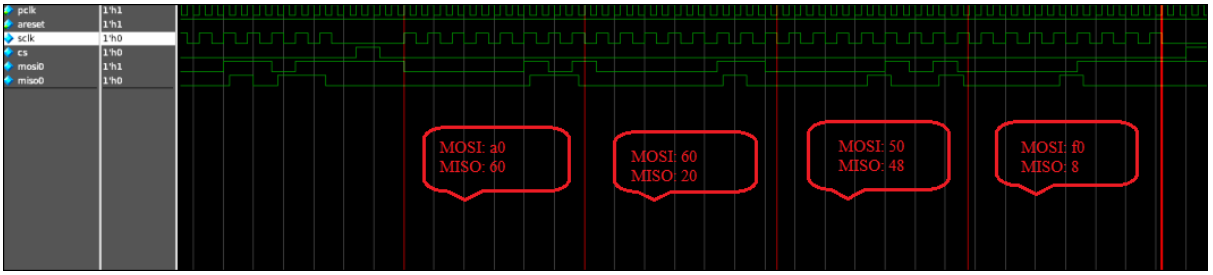


Fig 4.34 wave form for 32bit data transfer (2nd transaction)

Fig 4.34 shows the waveform for the 2nd transaction with a whole 32 bit full duplex transfer.

Chapter 5

References

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