2021

SPI-AVIP

USER GUIDE

Contents

List of Tables

List of Figures

4.2	Master_agent_config values	
4.3	Slave_agent_config values	
4.4	Env_config values	
4.5	Transfer size for mosi and miso	
4.6	Master_tx values	
4.7	Slave_tx values	
4.8	Converted data of master req	
4.9	Converted data of slave req	
4.10	Master bfm_struct data	
4.11	Slave bfm_struct data	
4.12	Master_driver_bfm values	
4.13	Slave_driver_bfm values	
4.14	Master_monitor values	
4.15	Slave_monitor values	
4.16	Scoreboard_checks	
4.17	Coverage for master	
4.18	Coverage for slave	
4.19	Master and slave coverage	
4.20	Instance of cover group	
4.21	Master_coverage coverpoint	
4.22	Master_coverage crosses coverpoints	
4.23	Slave_coverage coverpoint	
4.24	Slave_coverage crosses coverpoints	
4.25	Log file for assertion test	
4.26	Assertion errors	
4.27	Assertion property-1	

Chapter 1

Introduction

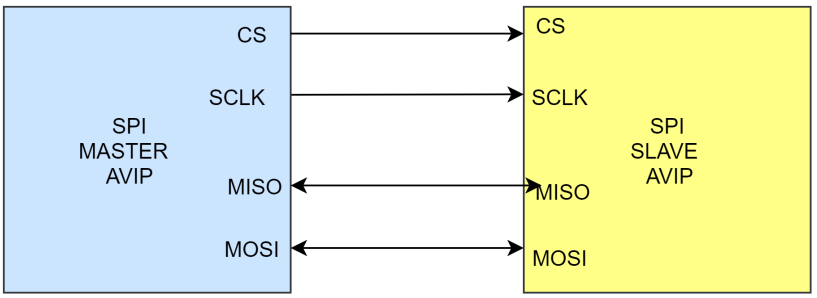


Fig: 1.1 simple spi avip

Master avip can communicate with the slave avip via spi 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 Spi master avip

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

Spi interface has the following interface pin level signals :

1. pclk
2. areset_n
3. sclk
4. cs_n
5. mosi0
6. mosi1
7. mosi2
8. mosi3
9. miso0
10. miso1
11. miso2
12. miso3

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

1.4 spi_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 spi interface

- ❖ If you want to add any file in the project, please add the file or folder in spi_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 spi_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 spi_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/spi_interface
```

Fig: 1.2 Directories included in spi_compile.f file

b. Packages included:

```
../../src/globals/spi_globals_pkg.sv
../../src/hvl_top/master/spi_master_pkg.sv
../../src/hvl_top/slave/spi_slave_pkg.sv
../../src/hvl_top/test/sequences/master_sequences/spi_master_seq_pkg.sv
../../src/hvl_top/test/sequences/slave_sequences/spi_slave_seq_pkg.sv
../../src/hvl_top/env/spi_env_pkg.sv
../../src/hvl_top/test/virtual_sequences/spi_virtual_seq_pkg.sv
../../src/hvl_top/test/test_pkg.sv
```

Fig: 1.3 Packages included in spi_compile.f file

c. Static files included :

```
../../src/hdl_top/spi_interface/spi_if.sv
../../src/hdl_top/master_agent_bfm/master_driver_bfm.sv
../../src/hdl_top/master_agent_bfm/master_monitor_bfm.sv
../../src/hdl_top/master_agent_bfm/master_agent_bfm.sv
../../src/hdl_top/slave_agent_bfm/slave_driver_bfm.sv
../../src/hdl_top/slave_agent_bfm/slave_monitor_bfm.sv
../../src/hdl_top/slave_agent_bfm/slave_agent_bfm.sv
../../src/hdl_top/hdl_top.sv
../../src/hvl_top/hvl_top.sv
../../src/hdl_top/master_assertions.sv
../../src/hdl_top/slave_assertions.sv
```

Fig: 1.4 static files included in spi_compile.f file

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

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

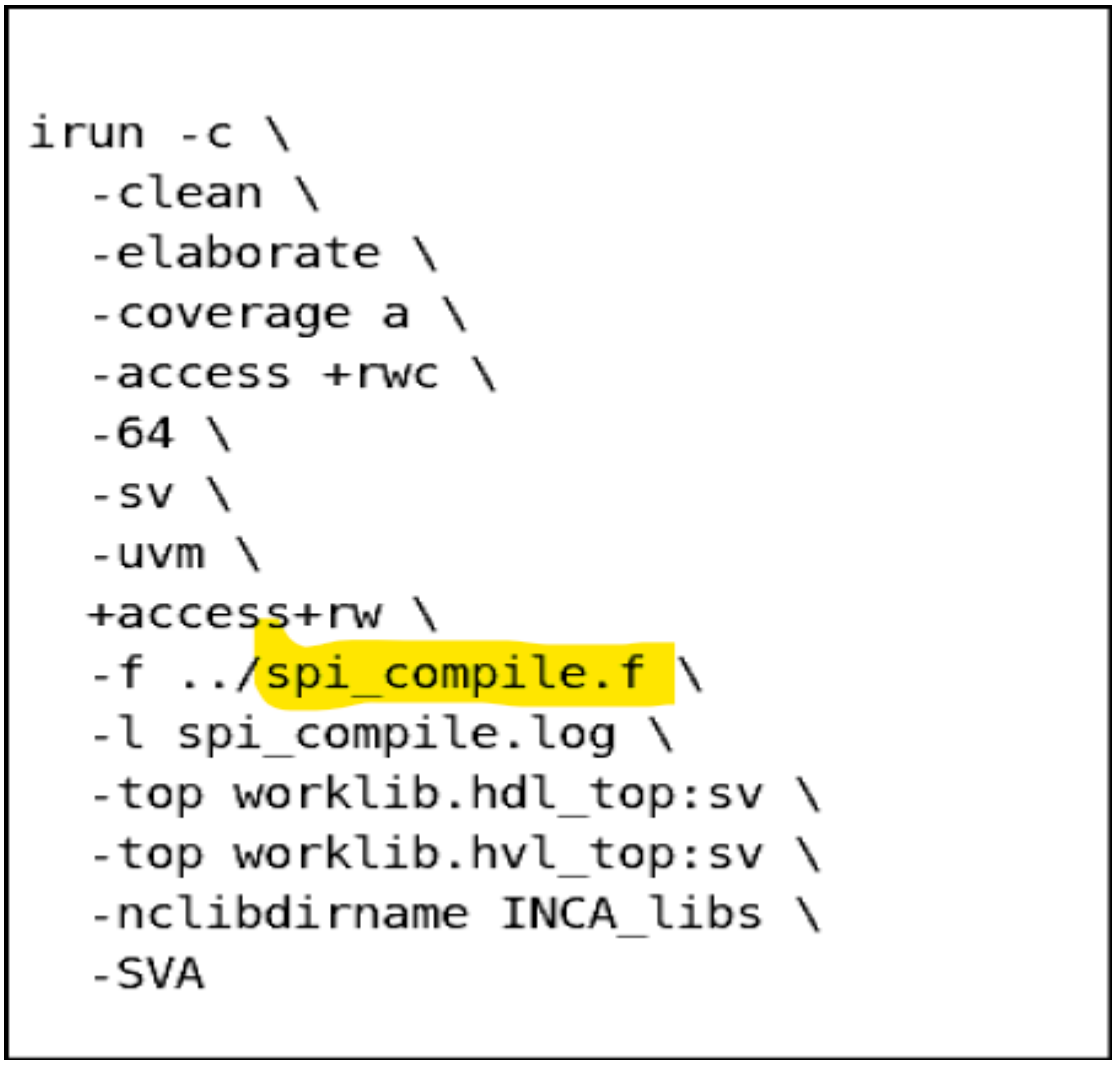


Fig 1.6 spi_compile.f used in makefile for cadence tool

Chapter 2

Architecture

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

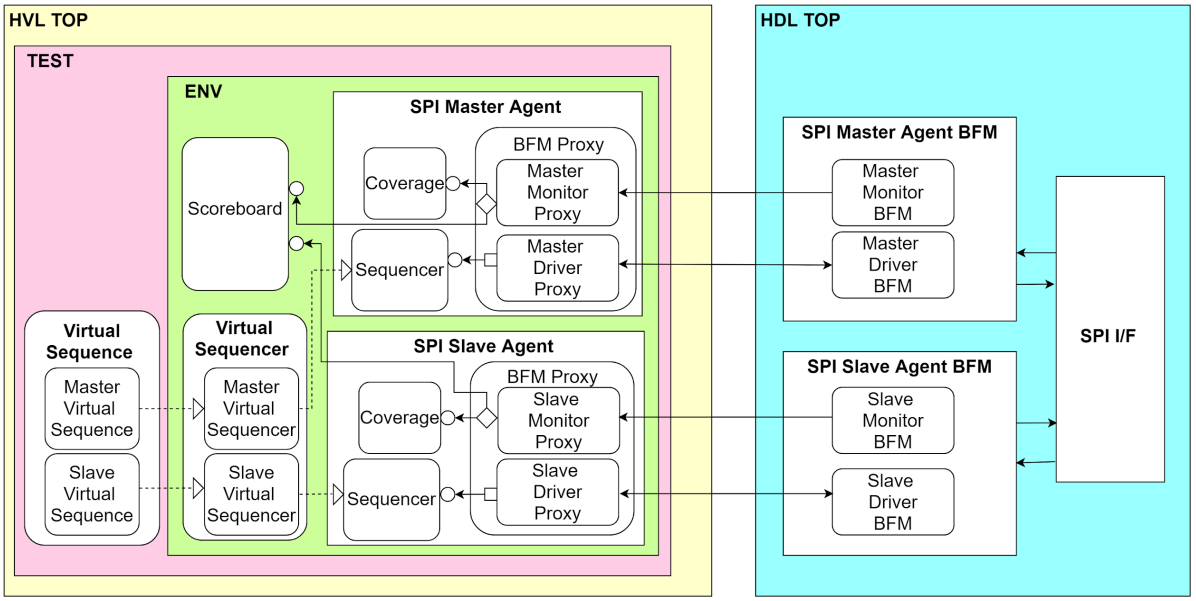


Fig 2.1 Spi avip Architecture

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 SPI 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.

We have the 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 Testcases

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 spi_avip repository in the terminal

   find the spi_avip GitHub repository here

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

   cd spi_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 spi_avip repository

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

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

2. To view the usage for running testcases, type the command

   make

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

```
[vandanam@HweServer questa_sim]$ make

------------------------------------------------------------------
------------------------- 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>

Example::
make regression testlist_name=spi_simple_fd_regression.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 spi_compile.log to view the compiled files
   gvim spi_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:

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

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

   Ex: *gvim spi_simple_fd_8b_test/spi_simple_fd_8b_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 spi_simple_fd_8b_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

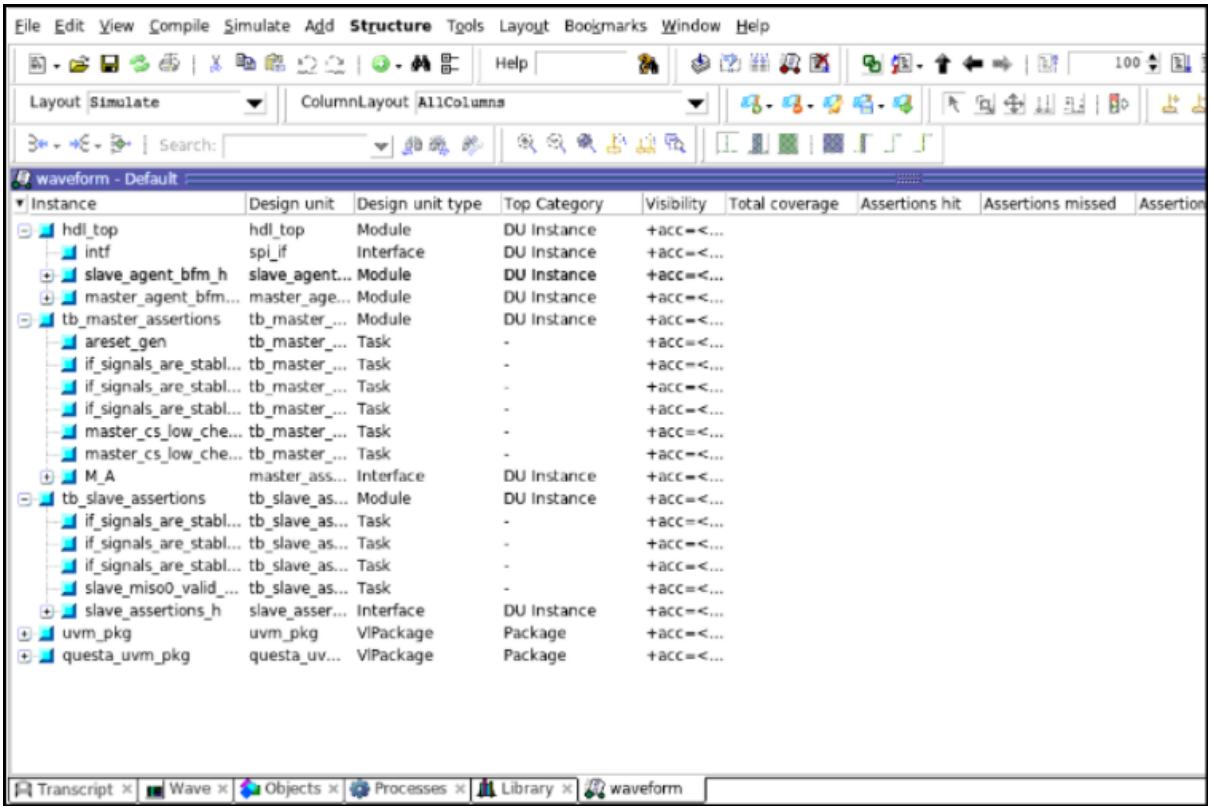


Fig 3.2 Questasim WLF window

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

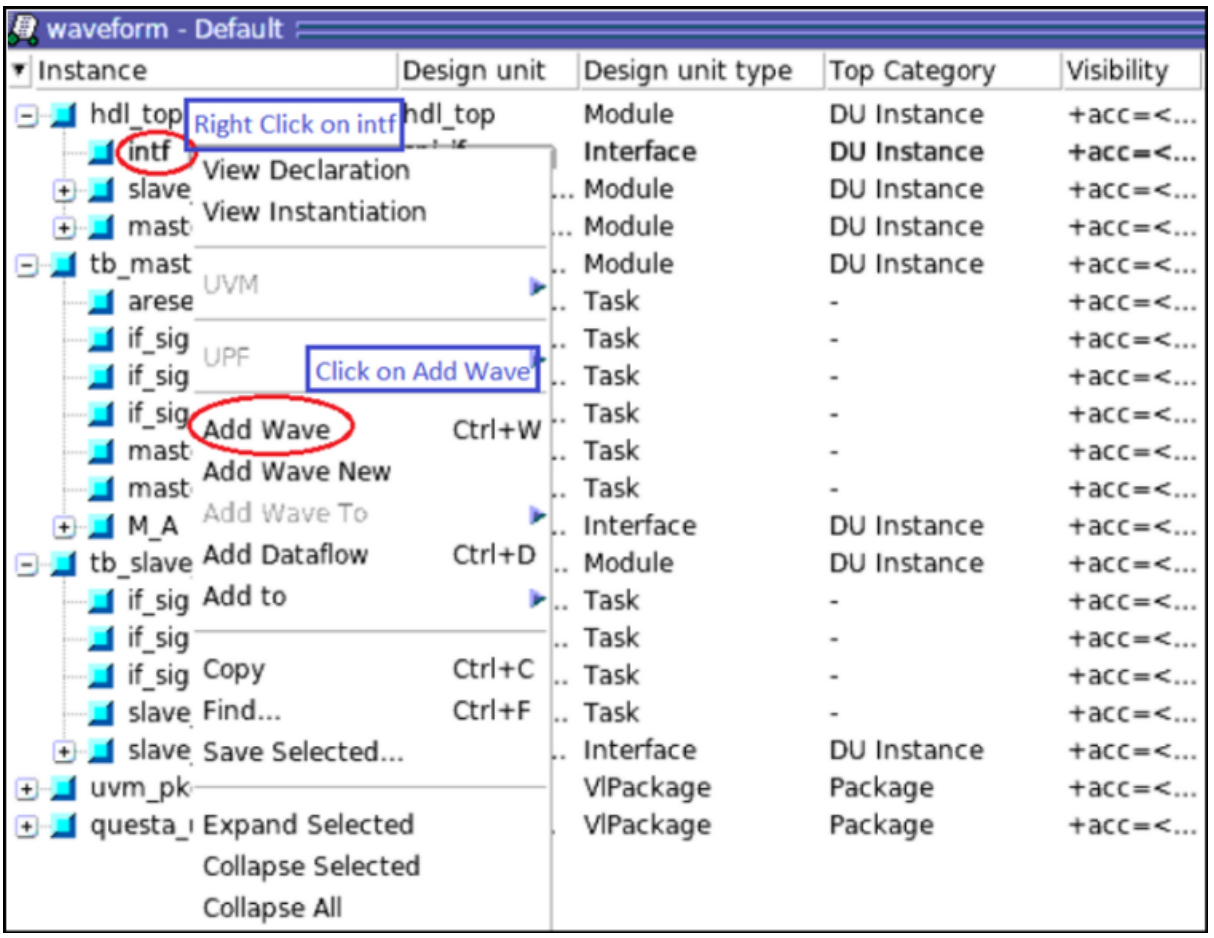


Fig 3.2 Screenshot of adding waves in wave window

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

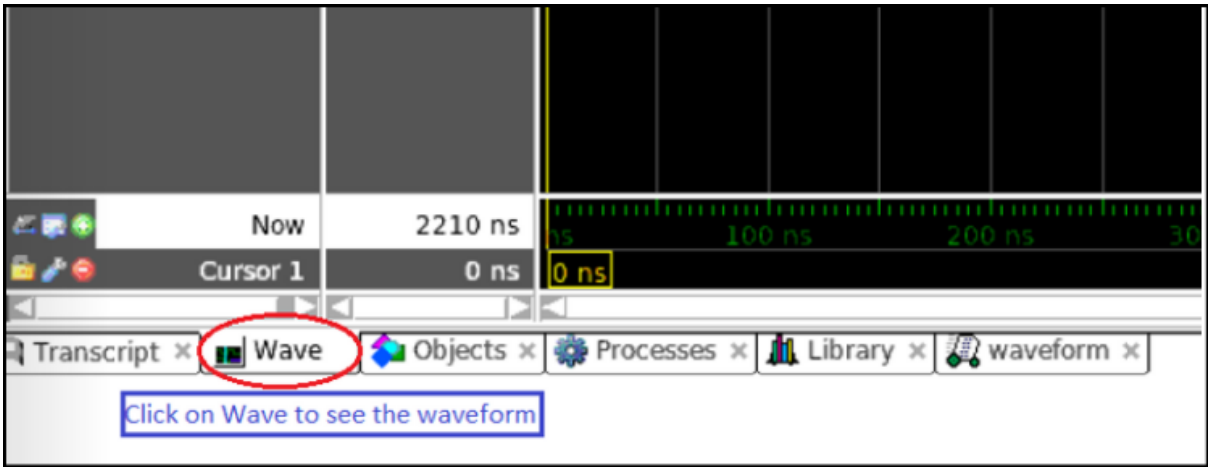


Fig 3.3 Wave window

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

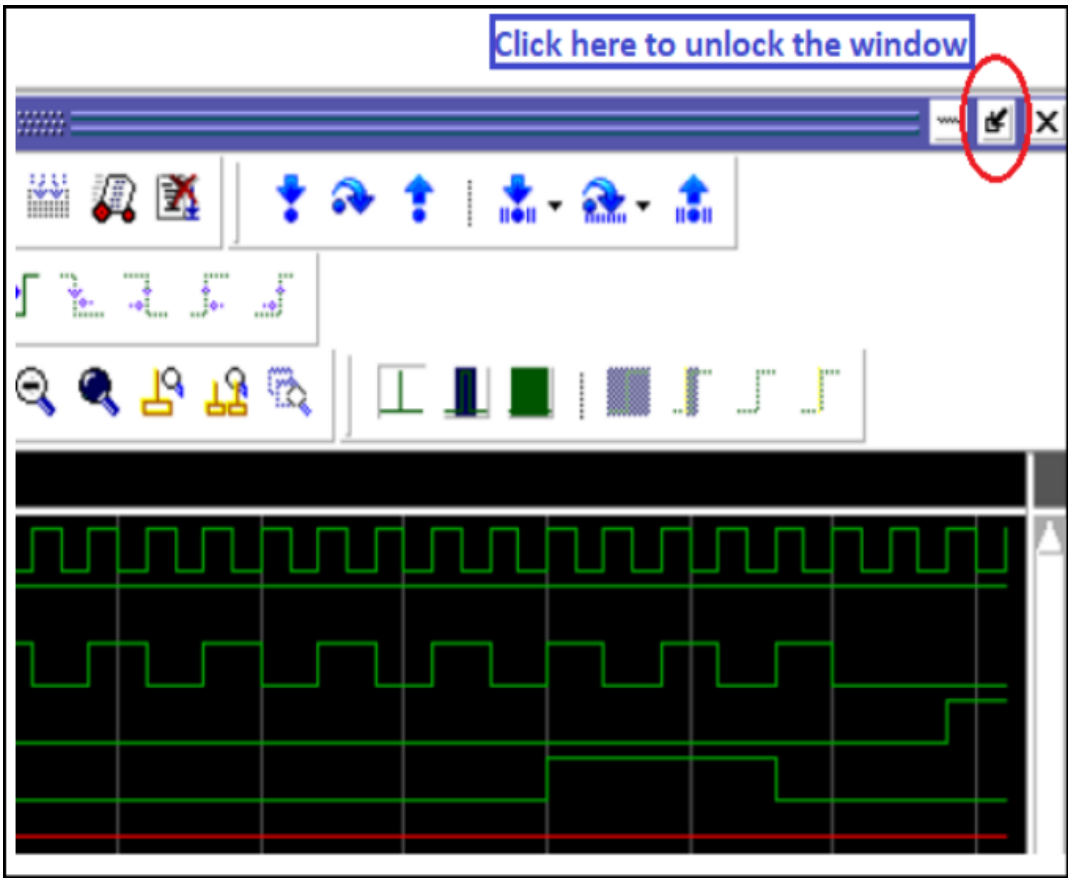


Fig 3.4 Screenshot of unlocking the wave window

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

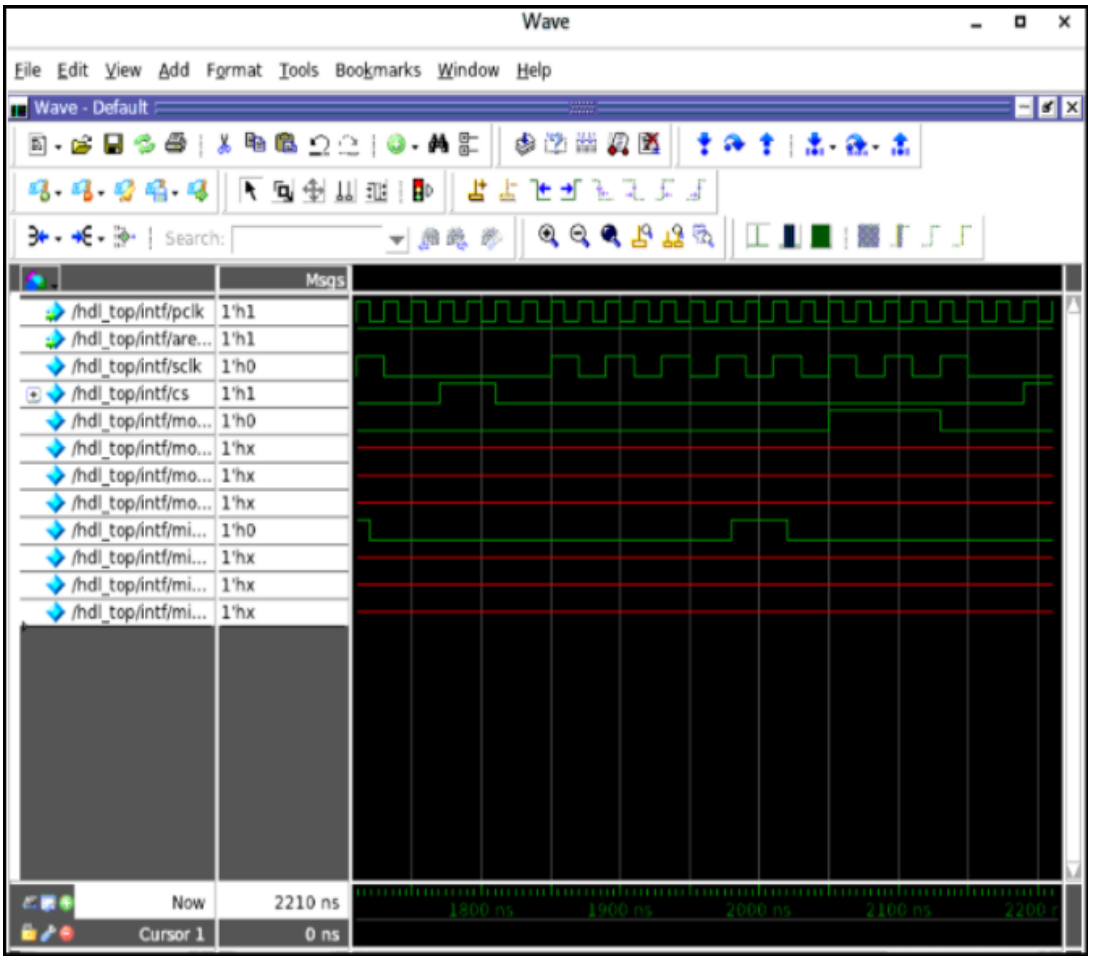


Fig 3.5 Screenshot of unlocked wave window with signals

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

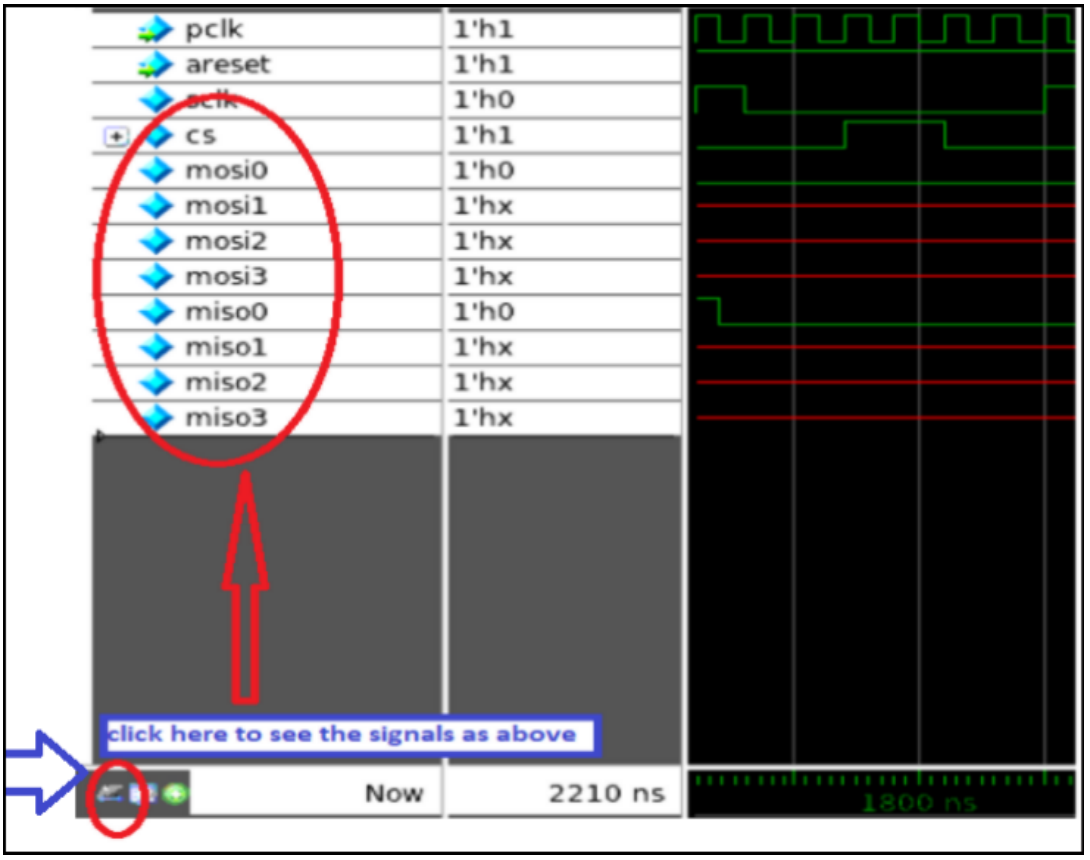


Fig 3.6 Screenshot showing way to see the name of signals

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=spi_simple_fd_regression.list*

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

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

   ls

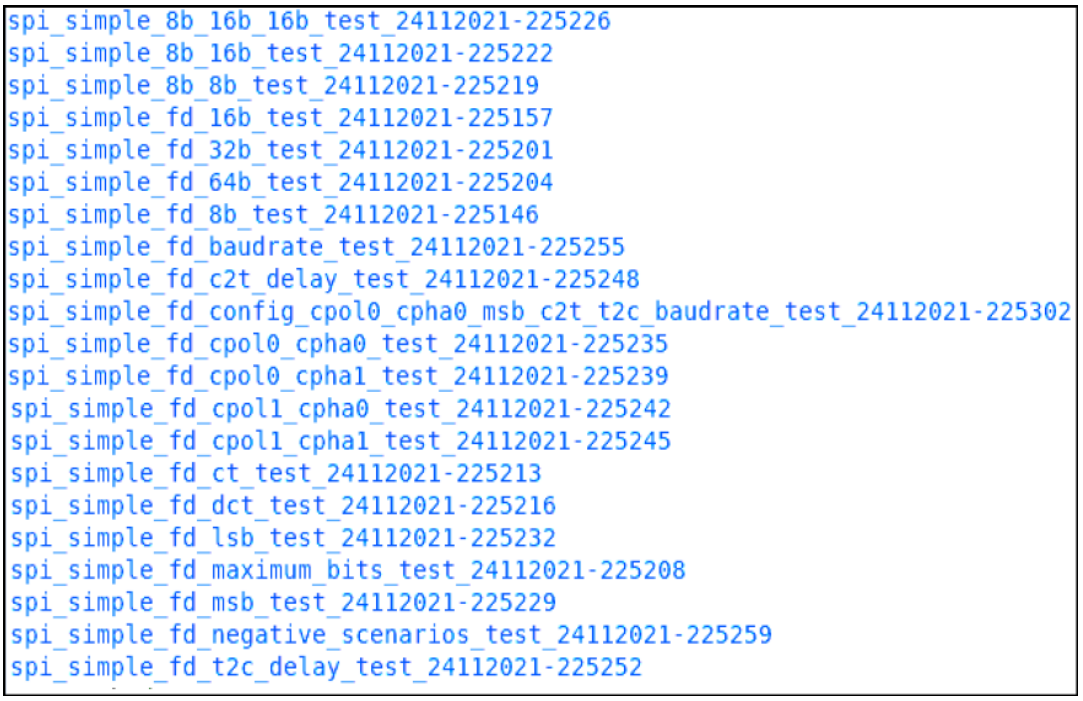


Fig 3.7 Files in questasim after the regression

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 *spi_simple_fd_cpol0_cpha1_test*

   Go inside the directory of interested testcase with the date

   spi_simple_fd_cpol0_cpha1_test_24112021-225239

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

   spi_simple_fd_cpol0_cpha1_test.log

   Path:

   spi_simple_fd_cpol0_cpha1_test_24112021-225239/spi_simple_fd_cpol0_cpha1_test. Log

3.2.4 Coverage

1. To see coverage

a. After simulating

For the individual test, use the command firefox

firefox spi_simple_fd_8b_test/html_cov_report/index.html &

Ex: *firefox spi_simple_fd_8b_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 *spi_simple_fd_cpol0_cpha1_test*

  Go inside the directory of interested testcase with the date

  spi_simple_fd_cpol0_cpha1_test_24112021-225239

  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

  firefox spi_simple_fd_cpol0_cpha1_test_24112021-225239/html_cov_report/covsummary.html

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

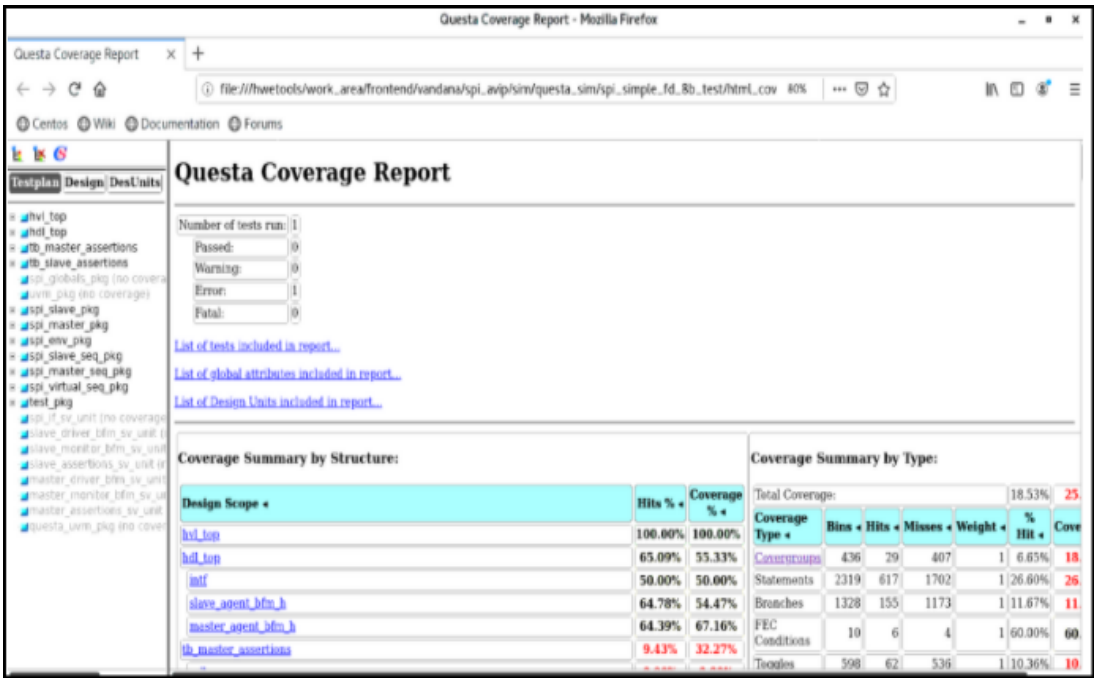
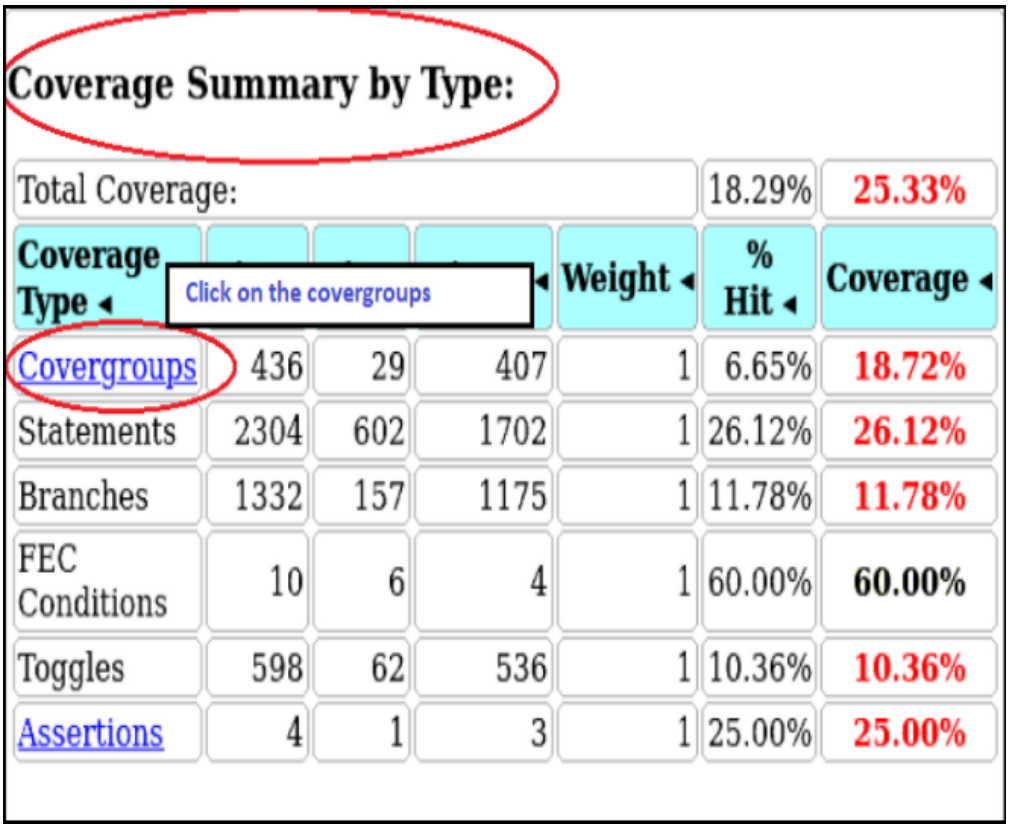


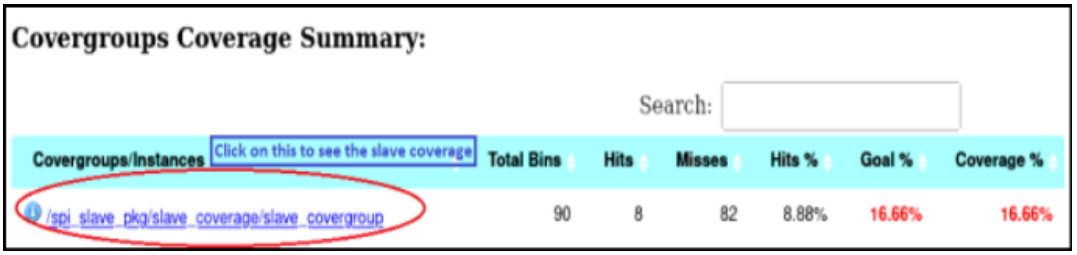
Fig 3.8 Coverage Report

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



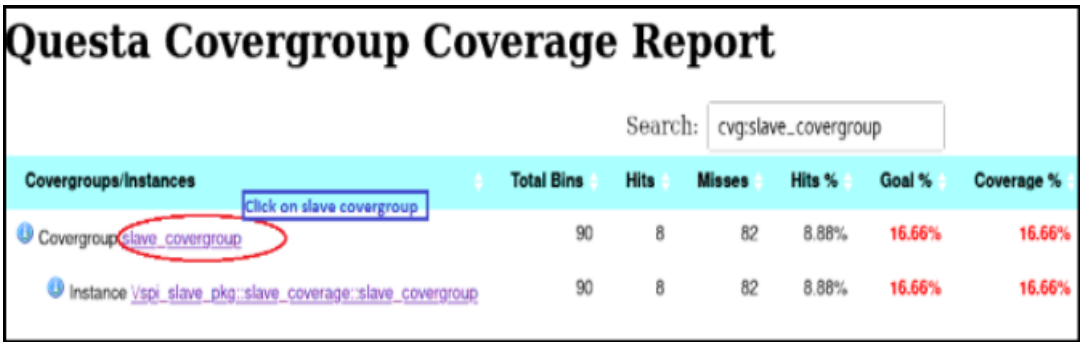
3.9 Screenshot of opening covergroups

4. After opening coevrgroup 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

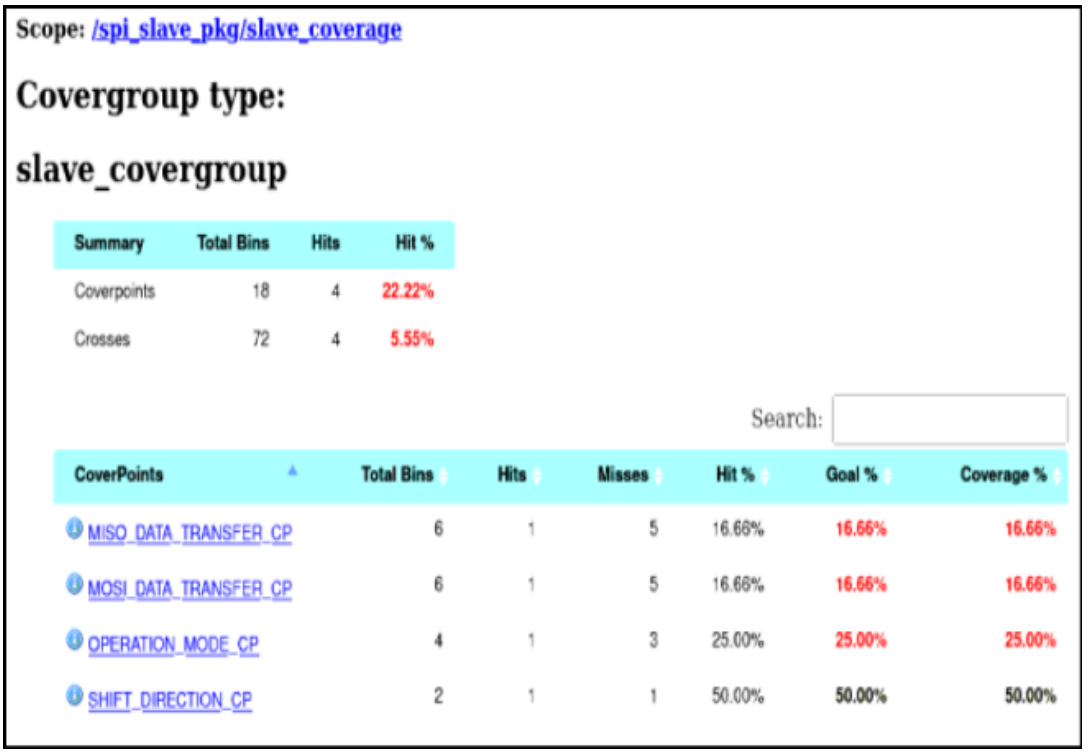


Fig 3.12 Screenshot of Coverpints and cross coverpoints

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

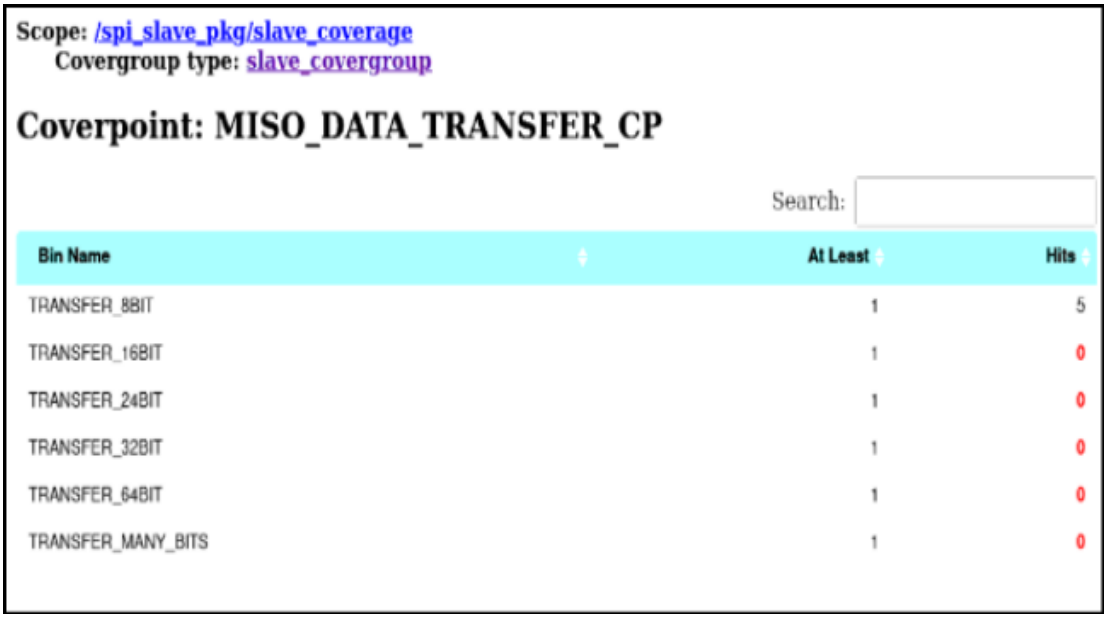


Fig. 3.13 MISO_DATA_TRANSFER_CP coverpoint report

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 spi_avip/sim/cadence_sim

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

2. To view the usage for running testcases, 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 spi_compile.log to view the compiled files
   vim spi_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
   spi_avip/src/hvl_top/testlists/spi_simple_fd_regression.list

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

   Ex: *gvim spi_simple_fd_8b_test/spi_simple_fd_8b_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
```

Fig 3.15 Simulation report in cadence

3. To view waveform type the command

   simvision waves.shm/

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

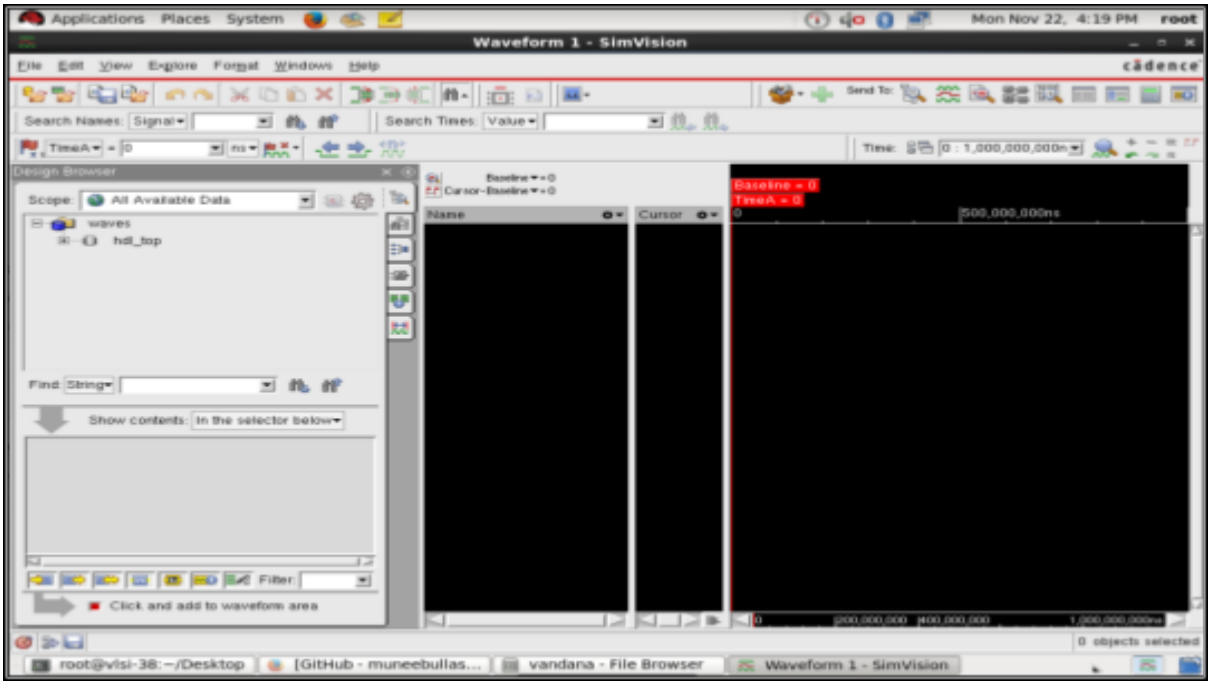


Fig 3.16 Cadence waveform window

5. 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

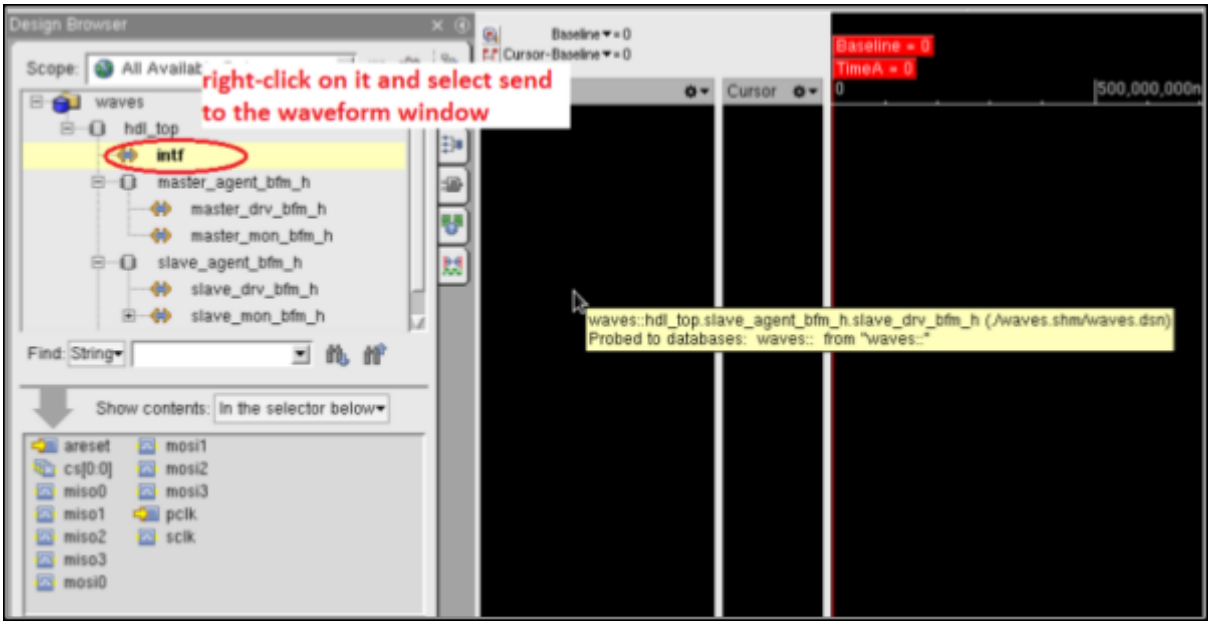


Fig 3.17

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

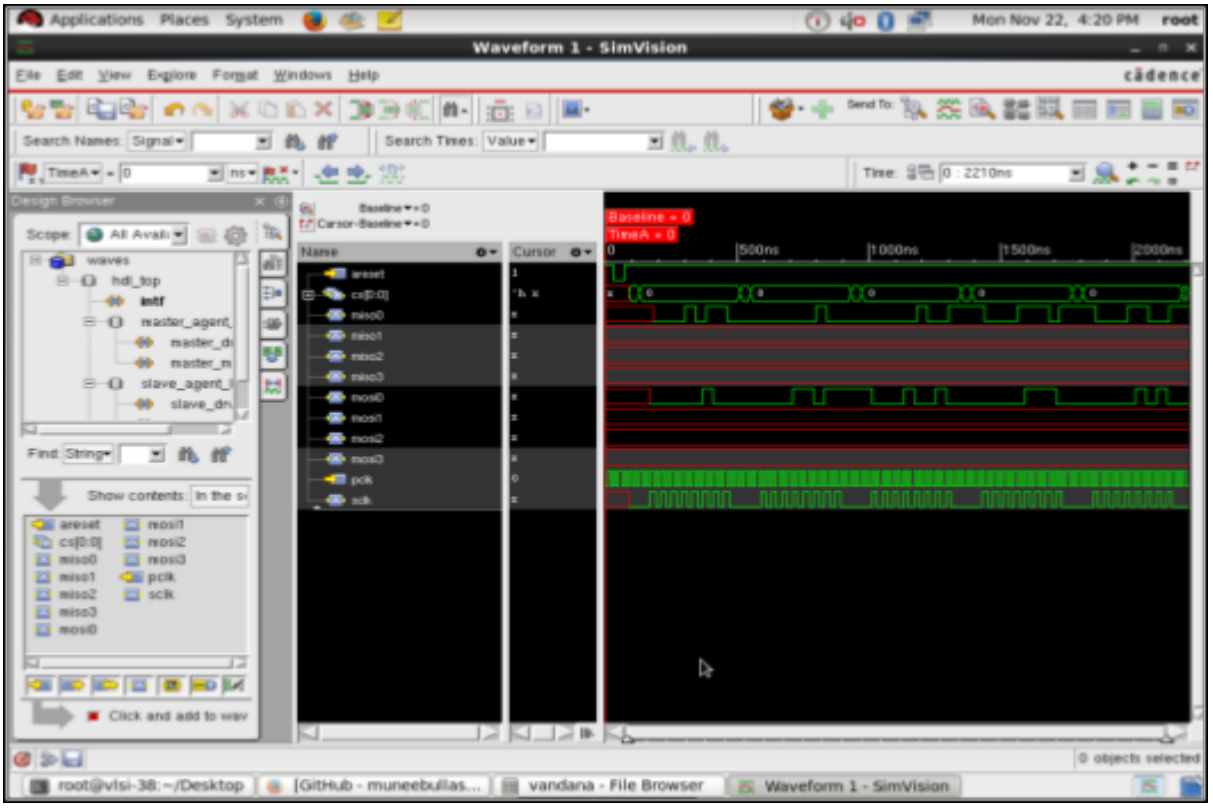


Fig 3.18

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

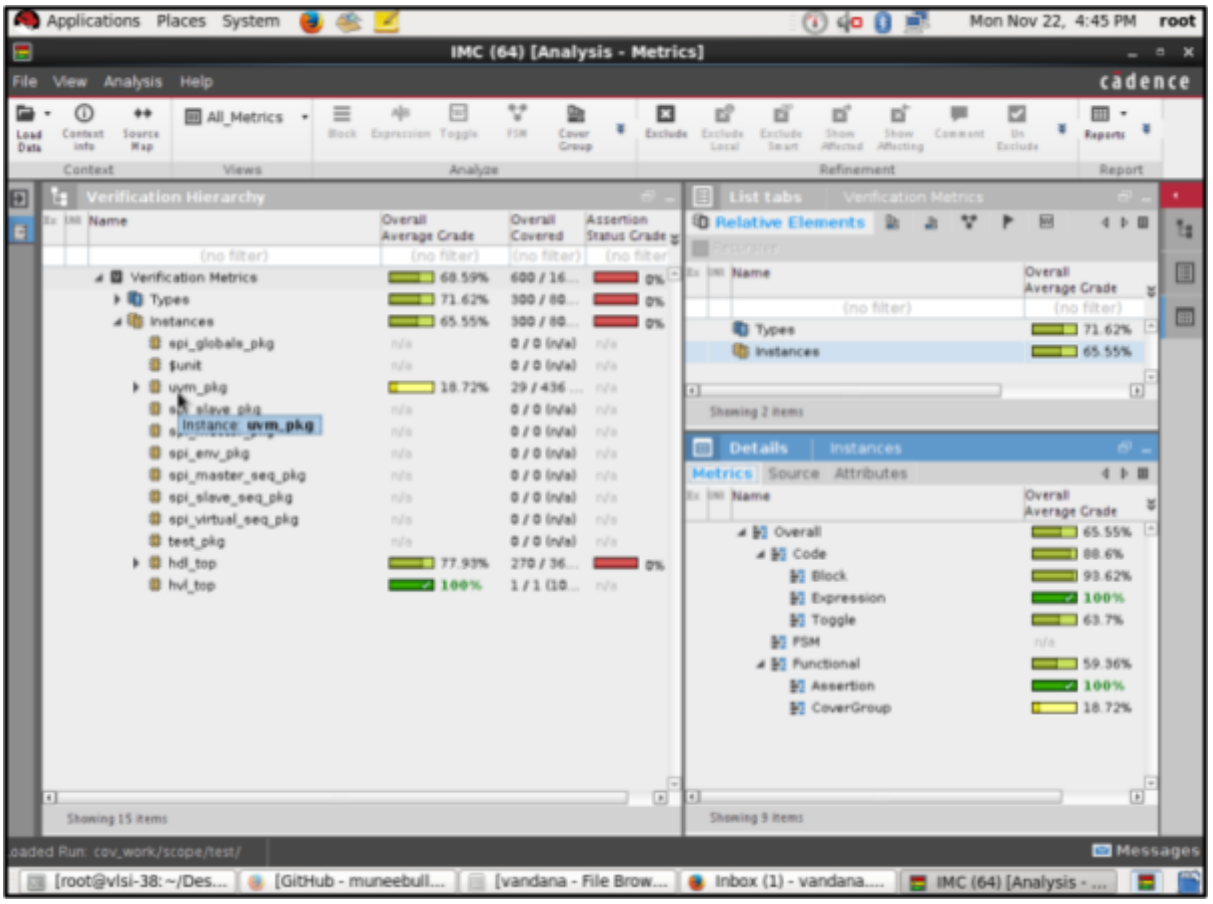


Fig 3.19

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

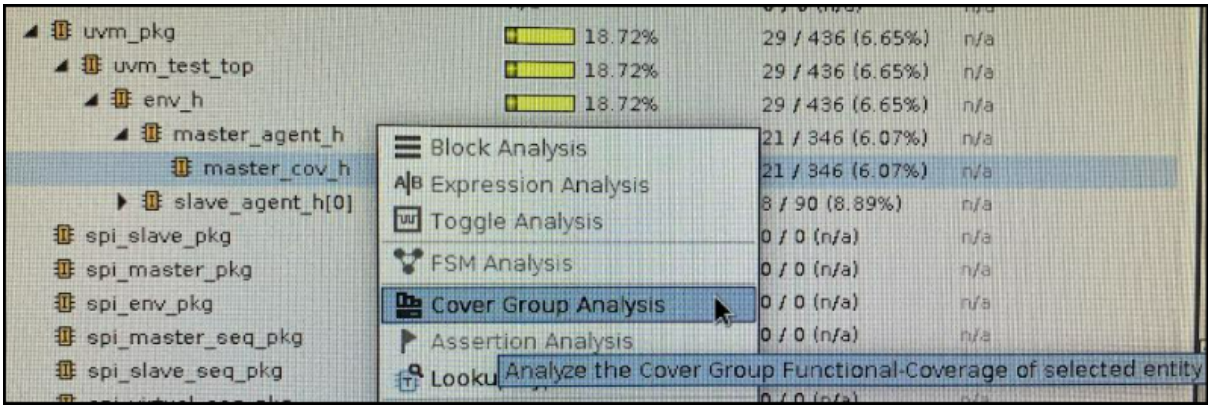
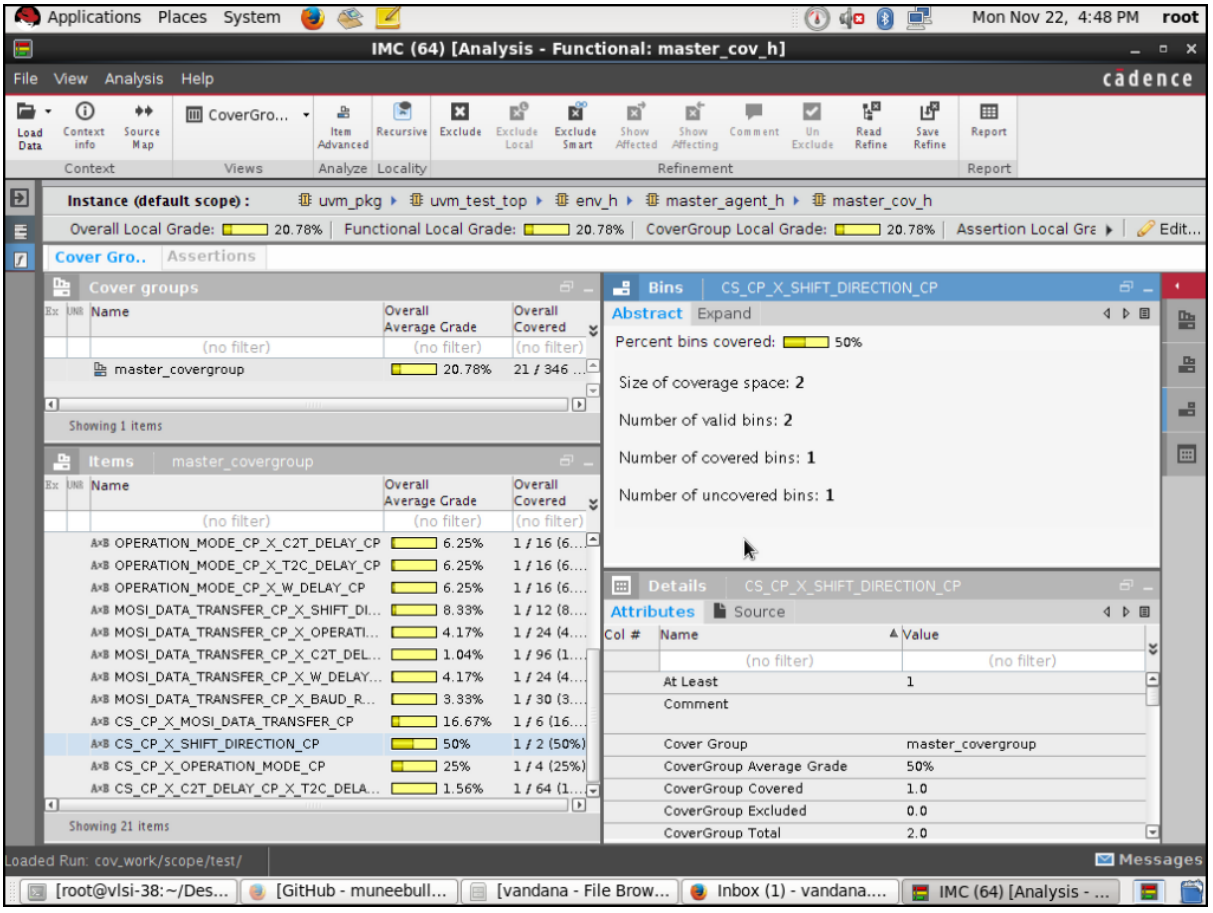
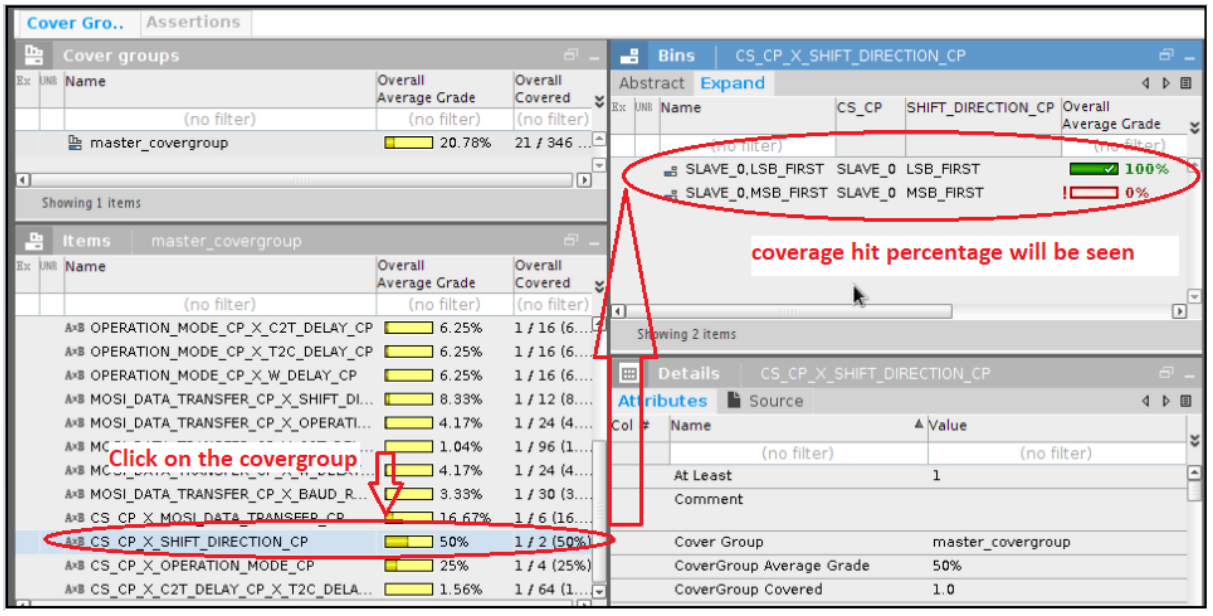


Fig 3.20

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 **spi_simple_fd_32b_test** has been used as an example test case in order to show the below debugging flow of the spi protocol and all the info's have been runned using **UVM_HIGH** verbosity

4.1 SPI 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.

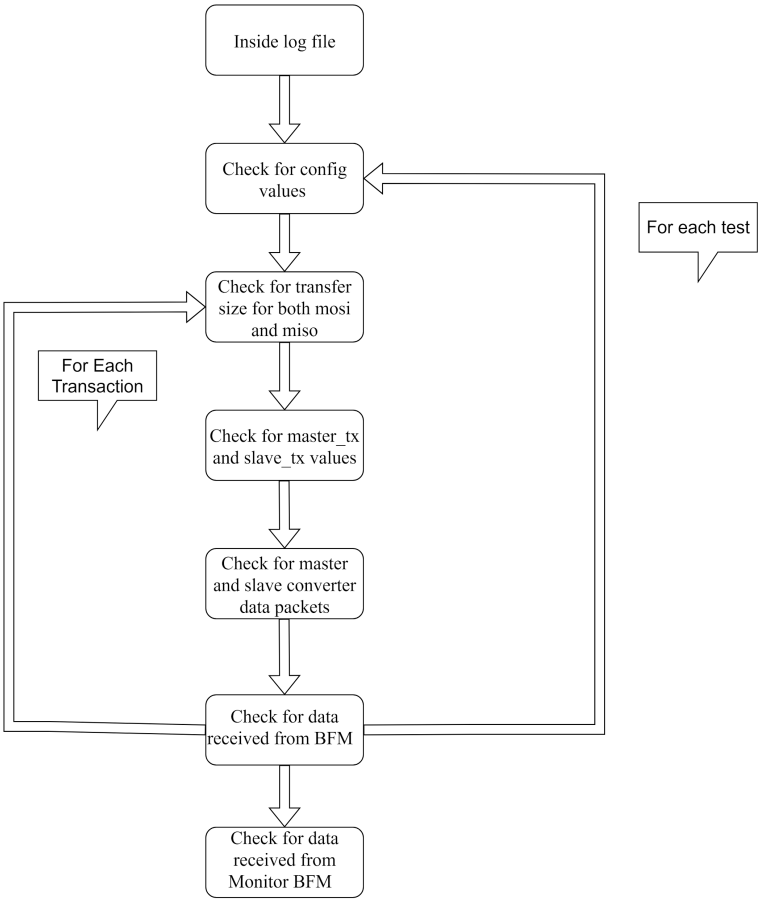


Fig 4.1 Debugging flow

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

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.
c2tdelay	which briefs about the delay between the chip select low to the transfer start and it should not equal zero.
t2cdelay	which briefs about the delay between the completion of transfer to the chip select High and it should not equal zero.
wdelay	which tells about the delay between the two transfers and it should not equal zero.
Primary and secondary prescaler	which are used to set the baud rate
Baud rate divisor	which basically tells at what rate the data transfer is happening.
has_coverage	Which indicates the coverage connection

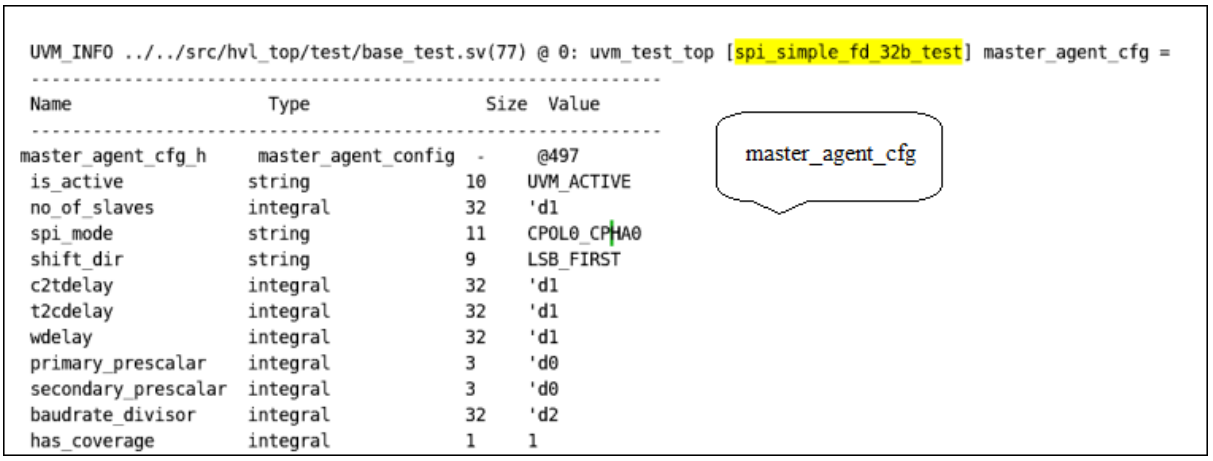


Fig 4.2 master_agent_config values

Figure 4.2 shows the different config values that has 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

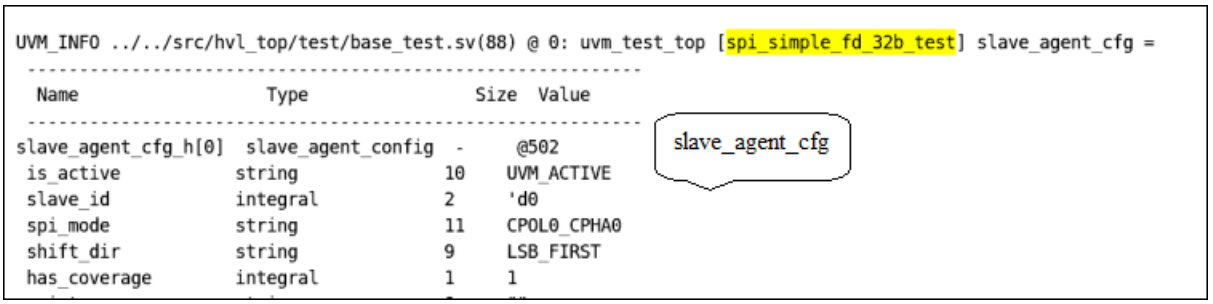


Fig 4.3 slave_agent_config values

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/base_test.sv(93) @ 0: uvm_test_top [spi_simple_fd_32b_test] env_cfg =
  ----------------------------------------
Name                Type        Size  Value
----------------------------------------
env_cfg_h           env_config  -     @496
  has_scoreboard    integral    1     1
  has_virtual_sqr   integral    1     1
  no_of_slaves      integral    32    'h1
----------------------------------------
```

env_cfg

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:

SPI follows one specific rule related to the transfer size of mosi and miso data.

Always the no of bits are bytes transferred by the mosi should be the same as no of bits or bytes transferred by miso.

```
# UVM_INFO ../../src/hvl_top/master//master_spi_seq_item_converter.sv(47) @ 120: reporter
[master_seq_item_conv_class] no_of_mosi_bits_transfer =
#  32
```

```
# UVM_INFO ../../src/hvl_top/slave/slave_spi_seq_item_converter.sv(47) @ 140: reporter
[slave_seq_item_conv_class] no_of_miso_bits_transfer =
#  32
```

Fig 4.5 Transfer size for mosi and miso

Figure 4.5 shows the transfer size for mosi and miso signals with the data to be driven on miso and sampled at mosi.

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 as well from slave_tx class

Initially check for the idle state of the transaction.(Ex: In this case cs = 1 and sclk = cpol). Once the chip select goes low, check the size of the transaction and it should be the same as no of mosi/miso bits to transfer.

```
#
#UVM_INFO ../../src/hvl_top/test/sequences/master_sequences//spi_fd_32b_master_seq.sv(49) @ 110: uvm_test_top.env_h.master_agent_h.
# master_seqr_h@@spi_fd_32b_master_seq_h [spi_fd_32b_master_seq] master seq =
# -----------------------------------------------------------------------------------------------------------
# Name                             Type       Size  Value
# -----------------------------------------------------------------------------------------------------------
# req                              master_tx  -     @1353
#   begin_time                     time       64    110
#   depth                          int        32    'd2
#   parent sequence (name)         string     23    spi_fd_32b_master_seq_h
#   parent sequence (full name)    string     71    uvm_test_top.env_h.master_agent_h.master_seqr_h.spi_fd_32b_master_seq_h
#   sequencer                      string     47    uvm_test_top.env_h.master_agent_h.master_seqr_h
#   cs                             integral   2     'b0
#   master_out_slave_in[0]         integral   8     'h58
#   master_out_slave_in[1]         integral   8     'h48
#   master_out_slave_in[2]         integral   8     'hb8
#   master_out_slave_in[3]         integral   8     'hd8
# -----------------------------------------------------------------------------------------------------------
#
```

Fig 4.6 master_tx values

```
#
# UVM_INFO ../../src/hvl_top/test/sequences/slave_sequences//spi_fd_32b_slave_seq.sv(42) @ 140: uvm_test_top.env_h.slave_agent_h[0].
# slave_seqr_h@@spi_fd_32b_slave_seq_h [spi_fd_32b_slave_seq] slave seq =
# -----------------------------------------------------------------------------------------------------------
# Name                             Type       Size  Value
# -----------------------------------------------------------------------------------------------------------
# req                              slave_tx   -     @1357
#   begin_time                     time       64    140
#   depth                          int        32    'd2
#   parent sequence (name)         string     22    spi_fd_32b_slave_seq_h
#   parent sequence (full name)    string     71    uvm_test_top.env_h.slave_agent_h[0].slave_seqr_h.spi_fd_32b_slave_seq_h
#   sequencer                      string     48    uvm_test_top.env_h.slave_agent_h[0].slave_seqr_h
#   master_in_slave_out[0]         integral   8     'h20
#   master_in_slave_out[1]         integral   8     'hd8
#   master_in_slave_out[2]         integral   8     'h78
#   master_in_slave_out[3]         integral   8     'h68
# -----------------------------------------------------------------------------------------------------------
#
```

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//master_spi_seq_item_converter.sv(65) @ 120: reporter [master_seq_item_conv_class]
# output_conv_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, 0,
#  0, 0, 216, 184, 72, 88}
```

Fig 4.8 converted data of master req

```
#
#
# UVM_INFO ../../src/hvl_top/slave/slave_spi_seq_item_converter.sv(53) @ 140: reporter [slave_seq_item_conv_class]
# output_conv_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, 104, 120, 216, 32}
#
```

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 the mosi signal and samples the miso depending on configurations of master and similarly slave driver BFM will drive the miso signal and samples the mosi depending on configurations of slave.

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

```
#
#
# UVM_INFO ../../src/hvl_top/master//master_driver_proxy.sv(192) @ 1490: uvm_test_top.env_h.master_agent_h.
# master_drv_proxy_h [master_driver_proxy] BFM STRUCT PACKET : ,
#  '{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, 0, 0, 0, 0, 0, 216, 184, 72, 88}, no_of_mosi_bits_transfer:32,
#  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, 104, 120, 216, 32}, no_of_miso_bits_transfer:32}
#
```

Fig 4.10 master bfm_struct data

```
#
#
# UVM_INFO ../../src/hvl_top/slave/slave_driver_proxy.sv(166) @ 1480: uvm_test_top.env_h.slave_agent_h[0].
# slave_drv_proxy_h [slave_driver_proxy] AFTER GETTING DATA FROM DRIVER BFM STRUCT DATA PACKET : ,
#  '{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, 0, 0, 0, 0, 0, 216, 184, 72, 88},
#  no_of_mosi_bits_transfer:32, 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, 104, 120, 216, 32}, no_of_miso_bits_transfer:32}
#
#
```

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.

```
#
#
# UVM_INFO ../../src/hvl_top/master//master_driver_proxy.sv(171) @ 1490: uvm_test_top.env_h.master_agent_h.master_drv_proxy_h
# [master_driver_proxy] Received packet from MASTER DRIVER BFM : ,
#  ------------------------------------------------
# Name                      Type       Size  Value
# ------------------------------------------------
# master_tx                 master_tx  -     @1506
#   cs                      integral   2     'b0
#   master_out_slave_in[0]  integral   8     'h58
#   master_out_slave_in[1]  integral   8     'h48
#   master_out_slave_in[2]  integral   8     'hb8
#   master_out_slave_in[3]  integral   8     'hd8
#   master_in_slave_out[0]  integral   8     'h20
#   master_in_slave_out[1]  integral   8     'hd8
#   master_in_slave_out[2]  integral   8     'h78
#   master_in_slave_out[3]  integral   8     'h68
# ------------------------------------------------
#
#
```

Fig 4.13 master_driver_bfm values

```
#
#
# UVM_INFO ../../src/hvl_top/slave/slave_driver_proxy.sv(138) @ 1480: uvm_test_top.env_h.slave_agent_h[0].slave_drv_proxy_h
# [slave_driver_proxy] Received packet from SLAVE DRIVER BFM : ,
#  ------------------------------------------------
# Name                      Type       Size  Value
# ------------------------------------------------
# slave_tx                  slave_tx   -     @1481
#   master_in_slave_out[0]  integral   8     'h20
#   master_in_slave_out[1]  integral   8     'hd8
#   master_in_slave_out[2]  integral   8     'h78
#   master_in_slave_out[3]  integral   8     'h68
#   master_out_slave_in[0]  integral   8     'h58
#   master_out_slave_in[1]  integral   8     'h48
#   master_out_slave_in[2]  integral   8     'hb8
#   master_out_slave_in[3]  integral   8     'hd8
# ------------------------------------------------
#
#
```

Fig 4.14 slave_driver_bfm values

Fig 4.13 and 4.14 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

```
#
# UVM_INFO ../../src/hvl_top/master//master_monitor_proxy.sv(161) @ 1480: uvm_test_top.env_h.master_agent_h.master_mon_proxy_h
# [master_monitor_proxy] Received packet from MASTER MONITOR BFM : ,
#  ------------------------------------------------
# Name                       Type       Size  Value
# ------------------------------------------------
# master_tx                  master_tx  -     @1485
#   cs                       integral   2     'b0
#   master_out_slave_in[0]   integral   8     'h58
#   master_out_slave_in[1]   integral   8     'h48
#   master_out_slave_in[2]   integral   8     'hb8
#   master_out_slave_in[3]   integral   8     'hd8
#   master_in_slave_out[0]   integral   8     'h20
#   master_in_slave_out[1]   integral   8     'hd8
#   master_in_slave_out[2]   integral   8     'h78
#   master_in_slave_out[3]   integral   8     'h68
# ------------------------------------------------
#
```

Fig 4.15 master_monitor values

```
#
#
# UVM_INFO ../../src/hvl_top/slave/slave_monitor_proxy.sv(257) @ 1480: uvm_test_top.env_h.slave_agent_h[0].slave_mon_proxy_h
# [slave_monitor_proxy] Received packet from SLAVE MONITOR BFM : ,
#  ------------------------------------------------
# Name                       Type       Size  Value
# ------------------------------------------------
# slave_tx                   slave_tx   -     @1493
#   master_in_slave_out[0]   integral   8     'h20
#   master_in_slave_out[1]   integral   8     'hd8
#   master_in_slave_out[2]   integral   8     'h78
#   master_in_slave_out[3]   integral   8     'h68
#   master_out_slave_in[0]   integral   8     'h58
#   master_out_slave_in[1]   integral   8     'h48
#   master_out_slave_in[2]   integral   8     'hb8
#   master_out_slave_in[3]   integral   8     'hd8
# ------------------------------------------------
#
```

Fig 4.16 slave_monitor values

4.2 Scoreboard Checks:

And finally we have scoreboard checks which basically compares the mosi data of master with the mosi data of slave and similarly miso data of slave with the miso data of master

```
#
#
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(120) @ 1480: uvm_test_top.env_h.scoreboard_h
# [spi_scoreboard] Size of MOSI data from Master and Slave is equal
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(138) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MOSI_DATA_MATCH] Master MOSI[0] = 'h58 and Slave MOSI[0] = 'h58
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(138) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MOSI_DATA_MATCH] Master MOSI[1] = 'h48 and Slave MOSI[1] = 'h48
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(138) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MOSI_DATA_MATCH] Master MOSI[2] = 'hb8 and Slave MOSI[2] = 'hb8
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(138) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MOSI_DATA_MATCH] Master MOSI[3] = 'hd8 and Slave MOSI[3] = 'hd8
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(147) @ 1480: uvm_test_top.env_h.scoreboard_h
# [spi_scoreboard] Size of MISO data from Master and Slave is equal
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(165) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MISO_DATA_MATCH] Slave MISO[0] = 'h20 and Master MISO[0] = 'h20
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(165) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MISO_DATA_MATCH] Slave MISO[1] = 'hd8 and Master MISO[1] = 'hd8
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(165) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MISO_DATA_MATCH] Slave MISO[2] = 'h78 and Master MISO[2] = 'h78
# UVM_INFO ../../src/hvl_top/env/spi_scoreboard.sv(165) @ 1480: uvm_test_top.env_h.scoreboard_h
# [SB_MISO_DATA_MATCH] Slave MISO[3] = 'h68 and Master MISO[3] = 'h68
#
#
```

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//master_coverage.sv(198) @ 7010: uvm_test_top.env_h.master_agent_h.master_cov_h
# [master_coverage] Master Agent Coverage = 20.78 %
#
#
```

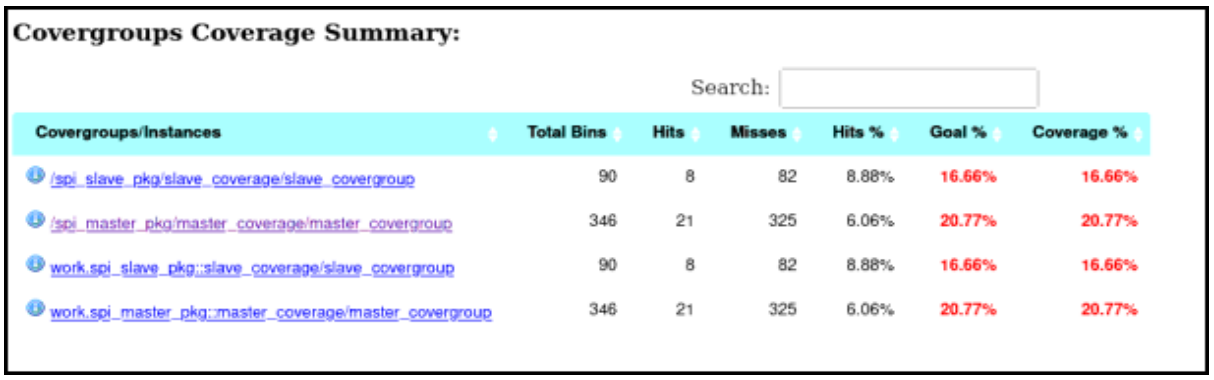
Fig 4.17 coverage for master

```
#
#
# UVM_INFO ../../src/hvl_top/slave/slave_coverage.sv(242) @ 7010: uvm_test_top.env_h.slave_agent_h[0].slave_cov_h
# [slave_coverage] Slave Agent Coverage = 16.67 %
#
#
```

Fig 4.18 coverage for slave

For individual bins checking goto the below html file
firefox spi_simple_fd_32b_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

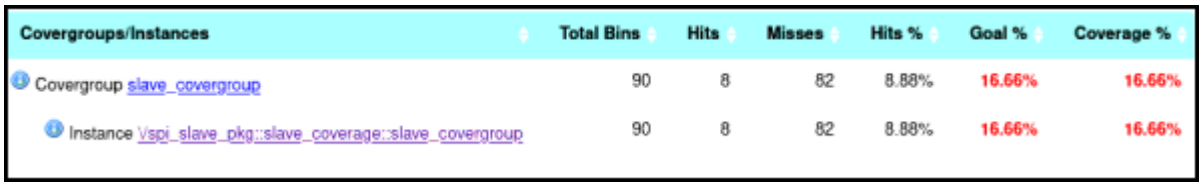


Covergroups Coverage Summary:

Search:

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
/spi_slave_pkg/slave_coverage/slave_covergroup	90	8	82	8.88%	16.66%	16.66%
/spi_master_pkg/master_coverage/master_covergroup	346	21	325	6.06%	20.77%	20.77%
work.spi_slave_pkg::slave_coverage/slave_covergroup	90	8	82	8.88%	16.66%	16.66%
work.spi_master_pkg::master_coverage/master_covergroup	346	21	325	6.06%	20.77%	20.77%

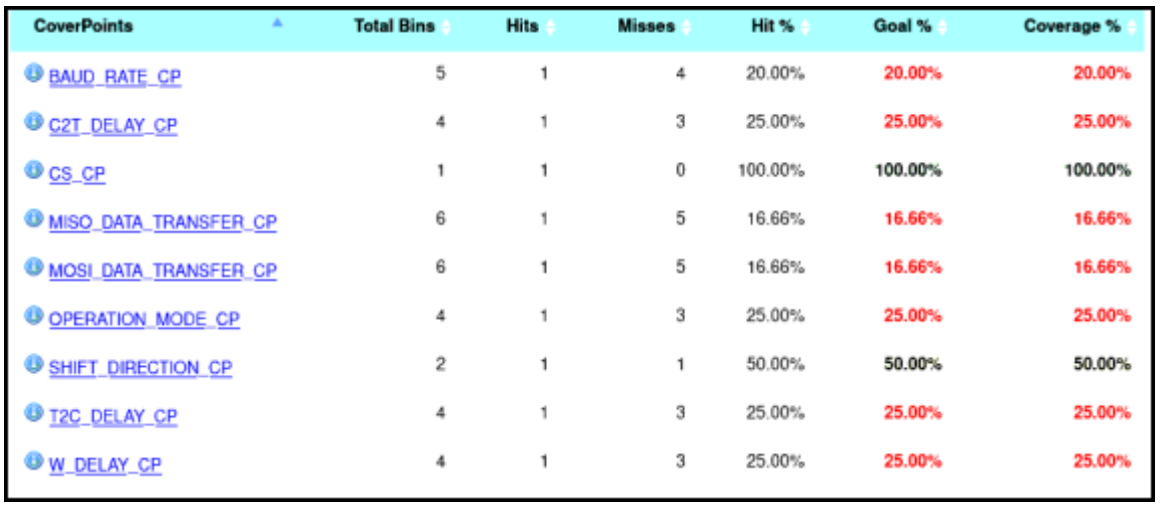
Fig 4.19 master and slave coverage



Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
Covergroup slave_covergroup	90	8	82	8.88%	16.66%	16.66%
Instance \/spi_slave_pkg::slave_coverage::slave_covergroup	90	8	82	8.88%	16.66%	16.66%

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.



CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
BAUD_RATE_CP	5	1	4	20.00%	20.00%	20.00%
C2T_DELAY_CP	4	1	3	25.00%	25.00%	25.00%
CS_CP	1	1	0	100.00%	100.00%	100.00%
MISO_DATA_TRANSFER_CP	6	1	5	16.66%	16.66%	16.66%
MOSI_DATA_TRANSFER_CP	6	1	5	16.66%	16.66%	16.66%
OPERATION_MODE_CP	4	1	3	25.00%	25.00%	25.00%
SHIFT_DIRECTION_CP	2	1	1	50.00%	50.00%	50.00%
T2C_DELAY_CP	4	1	3	25.00%	25.00%	25.00%
W_DELAY_CP	4	1	3	25.00%	25.00%	25.00%

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 %
CS_CP_X_C2T_DELAY_CP_X_T2C_DELAY_CP_X_W_DELAY_CP	64	1	63	1.56%	1.56%	1.56%
CS_CP_X_MOSI_DATA_TRANSFER_CP	6	1	5	16.66%	16.66%	16.66%
CS_CP_X_OPERATION_MODE_CP	4	1	3	25.00%	25.00%	25.00%
CS_CP_X_SHIFT_DIRECTION_CP	2	1	1	50.00%	50.00%	50.00%
MOSI_DATA_TRANSFER_CP_X_BAUD_RATE_CP	30	1	29	3.33%	3.33%	3.33%
MOSI_DATA_TRANSFER_CP_X_C2T_DELAY_CP_X_T2C_DELAY_CP	96	1	95	1.04%	1.04%	1.04%
MOSI_DATA_TRANSFER_CP_X_OPERATION_MODE_CP	24	1	23	4.16%	4.16%	4.16%
MOSI_DATA_TRANSFER_CP_X_SHIFT_DIRECTION_CP	12	1	11	8.33%	8.33%	8.33%
MOSI_DATA_TRANSFER_CP_X_W_DELAY_CP	24	1	23	4.16%	4.16%	4.16%
OPERATION_MODE_CP_X_C2T_DELAY_CP	16	1	15	6.25%	6.25%	6.25%
OPERATION_MODE_CP_X_T2C_DELAY_CP	16	1	15	6.25%	6.25%	6.25%
OPERATION_MODE_CP_X_W_DELAY_CP	16	1	15	6.25%	6.25%	6.25%

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	1	5	16.66%	16.66%	16.66%
MOSI_DATA_TRANSFER_CP	6	1	5	16.66%	16.66%	16.66%
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	1	23	4.16%	4.16%	4.16%
OPERATION_MODE_X_MASTER_OUT_SLAVE_IN	24	1	23	4.16%	4.16%	4.16%
SHIFT_DIRECTION_X_MASTER_IN_SLAVE_OUT	12	1	11	8.33%	8.33%	8.33%
SHIFT_DIRECTION_X_MASTER_OUT_SLAVE_IN	12	1	11	8.33%	8.33%	8.33%

Fig 4.24 slave_coverage crosses coverpoints

Figure 4.24 shows all the cross coverpoints included in slave coverage

4.4 Assertion Debug:

For the assertion debugging go to the assertion_base_test.log file

```
Testname: assertions_base_test

Log file path: assertions_base_test/assertions_base_test.log

Waveform: vsim -view assertions_base_test/waveform.wlf &
```

Fig 4.25 log file for assertion test

Inside the log file check for the assertion errors

```
# ** Error: Assertion error.
#    Time: 100 ns Started: 100 ns  Scope: tb_master_assertions.M_A.CPOL_1_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/master_assertions.sv Line: 149 Expr: ~$stable(mosi0)
# ** Error: Assertion error.
#    Time: 120 ns Started: 120 ns  Scope: tb_slave_assertions.slave_assertions_h.CPOL_0_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/slave_assertions.sv Line: 176 Expr: $stable(mosi0)
# ** Error: Assertion error.
#    Time: 140 ns Started: 140 ns  Scope: tb_master_assertions.M_A.CPOL_1_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/master_assertions.sv Line: 149 Expr: ~$stable(miso0)
# ** Error: Assertion error.
#    Time: 140 ns Started: 140 ns  Scope: tb_slave_assertions.slave_assertions_h.CPOL_0_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/slave_assertions.sv Line: 176 Expr: $stable(miso0)
# ** Error: Assertion error.
#    Time: 160 ns Started: 160 ns  Scope: tb_slave_assertions.slave_assertions_h.CPOL_0_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/slave_assertions.sv Line: 176 Expr: $stable(miso0)
# ** Error: Assertion error.
#    Time: 180 ns Started: 180 ns  Scope: tb_master_assertions.M_A.CPOL_1_CPHA_0_SIMPLE_SPI File:
#    ../../src/hdl_top/master_assertions.sv Line: 149 Expr: ~$stable(mosi0)
```

Fig 4.26 Assertion errors

Figure 4.26 shows the time stamp and the expression where the assertion errors are occuring.

```
property mode_of_cfg_cpol_1_cpha_0(logic mosi_local,logic miso_local);
  @(posedge sclk) disable iff(!areset)
  cpol==1 && cpha==0 |-> $stable(mosi_local) && $stable(miso_local);
endproperty: mode_of_cfg_cpol_1_cpha_0
`ifdef SIMPLE_SPI
  CPOL_1_CPHA_0_SIMPLE_SPI: assert property (mode_of_cfg_cpol_1_cpha_0(mosi0,miso0));
`endif
```

Fig 4.27 Assertion property(1)

Figure 4.27 shows the assertion property where it was trying to check the mosi and miso data is stable are unstable at every posedge of sclk

```
property mode_of_cfg_cpol_1_cpha_0(logic mosi_local,logic miso_local);
  @(posedge sclk) disable iff(!areset)
  cpol==1 && cpha==0 |-> !$isunknown(mosi_local) && !$isunknown(miso_local);
endproperty: mode_of_cfg_cpol_1_cpha_0
`ifdef SIMPLE_SPI
  CPOL_1_CPHA_0_SIMPLE_SPI: assert property (mode_of_cfg_cpol_1_cpha_0(mosi0,miso0));
`endif
```

Fig 4.28 Assertion property(2)

Figure 4.28 shows the assertion property where it was trying to check the mosi and miso data is unknown are not at every posedge of sclk

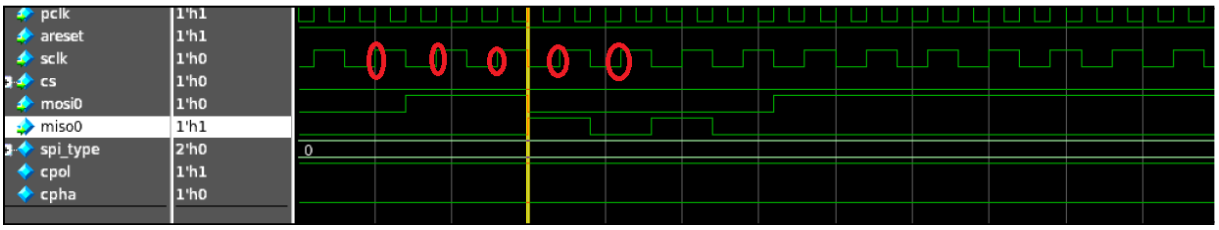


Fig 4.29 Assertion pass waveform

Fig. 4.29 shows the assertion waveform for the property shown in fig 4.28

4.5 Waveform Viewer

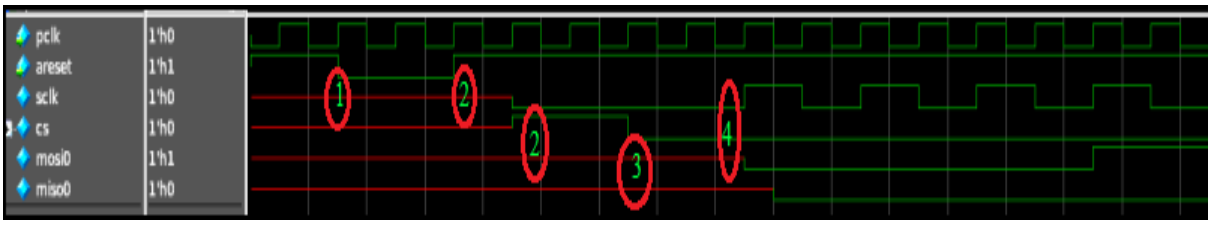


Fig 4.30 wave form for 32 bit full duplex test with initial conditions

The 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.

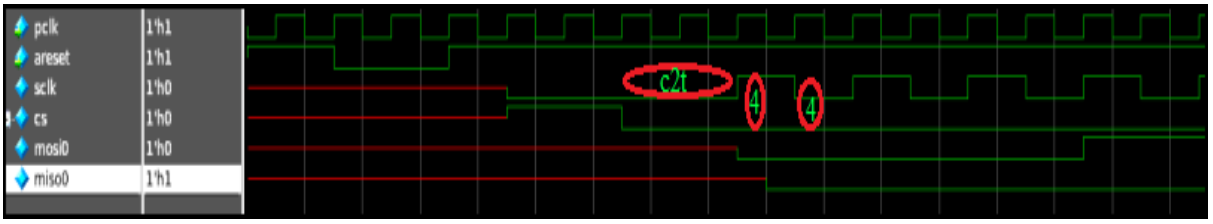


Fig 4.31 wave form for 32 bit full duplex test with c2tdelay

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.

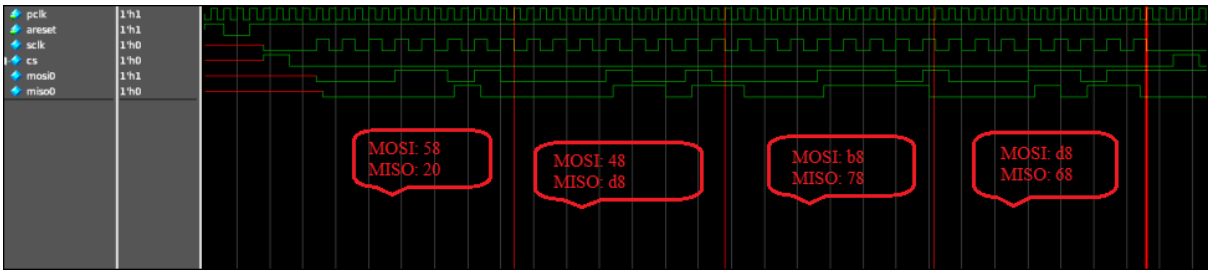


Fig 4.33 wave form for 32bit data transfer(1st transaction)

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

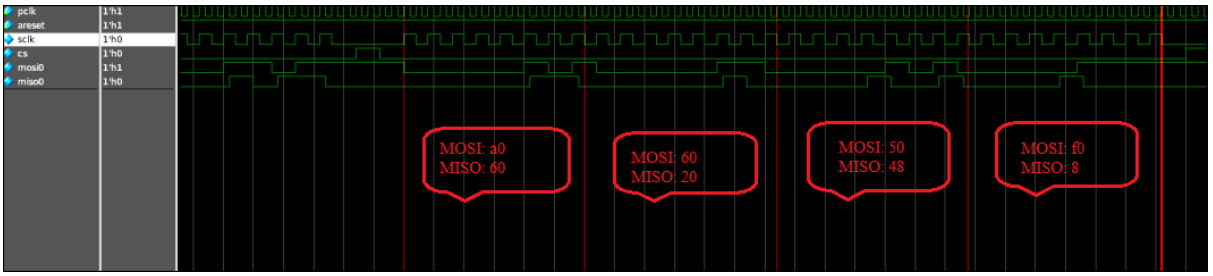


Fig 4.34 wave form for 32bit data transfer(2nd transaction)

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