

2021

PULPINO_SPI_MASTER_ IP_VERIFICATION

USER GUIDE

List of Tables	2
List of Figures	2
Chapter 1	5
Introduction	5
1.1 APB master avip	5
1.2 SPI slave avip	6
1.3 APB Interface and Spi Interface :	6
1.4 verif_compile.f file	7
1.5 rtl_compile.f file	8
1.6 Makefile Command	9
Chapter 2	9
Architecture	9
Chapter 3	12
Steps to run Test Cases	12
3.1 Git steps	12
3.2 Mentor's Questasim	13
3.2.1 Compilation	13
3.2.2 Simulation	13
3.2.3 Regression	17
3.2.4 Coverage	18
3.3 Cadence	22
3.3.1 Compilation	22
3.3.2 Simulation	22
3.3.3 Coverage	25
Chapter 4	28
Debug Tips	28
4.1 APB Debugging Flow	28
4.1.1 Check for Configuration Values	29
4.1.1(a) Master agent configurations	29
4.1.1(b) Slave agent configurations	29
4.1.1(c) Environment configuration	30
4.1.2 Check for transfer size	31
4.1.3 Check for transaction values	31
4.1.4 Check for master and slave converter data packets	32
4.1.5 Check for data received from BFM	33
4.1.6 Check for data received from monitor BFM	34
4.2 Scoreboard Checks	35

4.3 Coverage Debug	35
4.4 Assertion Debug	38
4.5 Waveform Viewer	40
Chapter 5	42
References	42

List of Tables

Table No	Table Name	Page No
Table 4.1	master configurations	27
Table 4.2	slave configurations	28
Table 4.3	environment configurations	29

List of Figures

Figure No	Name of the Figure	Page No
1.1	Simple APB AVIP	4
1.2	Directories included in verif_compile.f file	6
1.3	Packages included in verif_compile.f file	6

1.4	Static files included in verif_compile.f file	7
1.5	Packages and Static files included in rtl_compile.f file	7
1.5	rtl_compile.f used in makefile for questasim tool	7
1.6	verif_compile.f used in makefile for cadence tool	8
2.1	pulpino_spi_master_avip architecture	11
3.1	Usage of the make command	
3.2	Questasim WLF window	
3.3	Screenshot of adding waves in wave window	
3.4	Wave window	
3.5	Screenshot of unlocking the wave window	
3.6	Screenshot showing way to see the name of signals	
3.7	Files in questasim after the regression	
3.8	Coverage Report	
3.9	Screenshot of opening covergroups	
3.10	Screenshot of opening slave covergroup	
3.11	Shows way to open slave covergroup coverage report	
3.12	Screenshot of coverpoints and cross coverpoints	
3.13	PADDR_CP coverpoint report	
4.1	Debugging flow	
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	
4.28	Assertion property-2	
4.29	Assertion pass waveform	
4.30	Waveform for 32 bit full duplex test with initial conditions	
4.31	Waveform for 32 bit full duplex test with c2tdelay	
4.32	Wave form for 32 bit full duplex test with all the delays	
4.33	Wave form for 32bit data transfer(1st transaction)	
4.34	Wave form for 32bit data transfer(2nd transaction)	

Chapter 1

Introduction

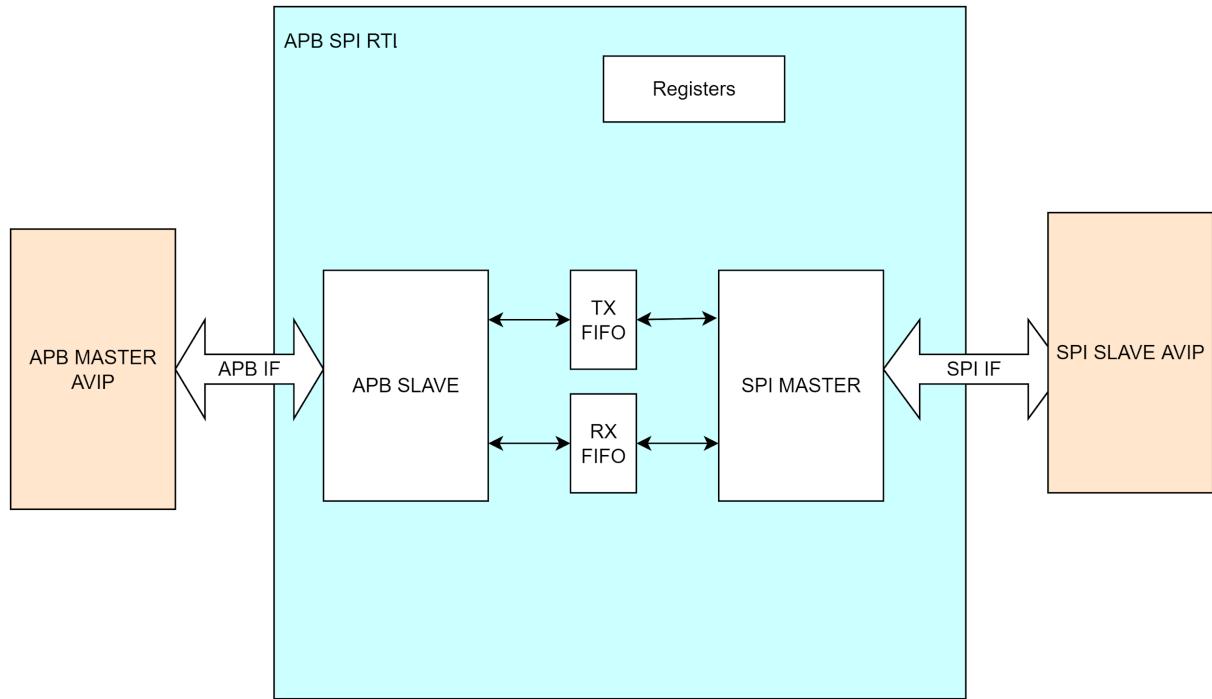


Fig: 1.1 Pulpino SPI Master IP Verification

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

1.1 APB master avip

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

To know more about inbound and outbound communication, please go through this link :
[Inbound and Outbound Communication](#)

1.2 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 APB Interface and Spi Interface :

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

1. pcclk
2. preset_n
3. paddr
4. pwrite
5. pwdata
6. pselx
7. prdata
8. pready
9. penable
10. pstrb
11. pprot
12. Pslverr
13. sclk
14. cs
15. cs1
16. cs2
17. cs3
18. mosi0
19. mosi1
20. mosi2
21. mosi3
22. miso0
23. miso1

24. miso2

25. miso3

To know more about the APB 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 APB 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/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/apb_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/apb_master_global_pkg.sv  
../../../../src/dv/globals/spi_slave_global_pkg.sv  
../../../../src/dv/globals/pulpino_spi_master_ip_global_pkg.sv  
../../../../src/dv/hvl_top/apb_master_agent/apb_master_pkg.sv  
../../../../src/dv/hvl_top/spi_slave_agent/spi_slave_pkg.sv  
../../../../src/dv/systemRDL/output/spi_masterDefines_pkg.svh  
../../../../src/dv/systemRDL/output/spi_master_uvm_pkg.sv  
../../../../src/dv/hvl_top/test/sequences/master_sequences/reg_sequences/apb_reg_seq_pkg.sv  
../../../../src/dv/hvl_top/test/sequences/master_sequences/apb_master_seq_pkg.sv  
../../../../src/dv/hvl_top/test/sequences/slave_sequences/spi_slave_seq_pkg.sv  
../../../../src/dv/hvl_top/env/pulpino_spi_master_ip_env_pkg.sv  
../../../../src/dv/hvl_top/test/virtual_sequences/pulpino_spi_master_ip_virtual_seq_pkg.sv  
../../../../src/dv/hvl_top/test/pulpino_spi_master_ip_test_pkg.sv
```

Fig: 1.3 Packages included in verif_compile.f file

c. Static files included :

```
../../../../src/dv/hdl_top/apb_if/apb_if.sv  
../../../../src/dv/hdl_top/spi_if/spi_if.sv  
../../../../src/dv/hdl_top/apb_master_agent_bfm/apb_master_agent_bfm.sv  
../../../../src/dv/hdl_top/apb_master_agent_bfm/apb_master_driver_bfm.sv  
../../../../src/dv/hdl_top/apb_master_agent_bfm/apb_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

1.5 rtl_compile.f file

This file contains the following things :

1. All the directories needed for the rtl
2. All the files needed for the rtl

```
+includer+../../src/rtl/spi_master/
../../src/rtl/spi_master/apb_spi_master.sv
../../src/rtl/spi_master/spi_master_apb_if.sv
../../src/rtl/spi_master/spi_master_clkgen.sv
../../src/rtl/spi_master/spi_master_controller.sv
../../src/rtl/spi_master/spi_master_fifo.sv
../../src/rtl/spi_master/spi_master_rx.sv
../../src/rtl/spi_master/spi_master_tx.sv
```

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

1.6 Makefile Command

In Makefile, we include the verif_compile.f file and rtl_compile.f file to compile all the files included.

Command used : *irun -f ./rtl_compile.f -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 \
        +fcov \
        -l pulpino_spi_master_ip_verification_compile.log \
        -f ./rtl_compile.f \
        -f ./verif_compile.f
```

Fig: 1.6 verif_compile.f used in makefile for questa_sim tool

Chapter 2

Architecture

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

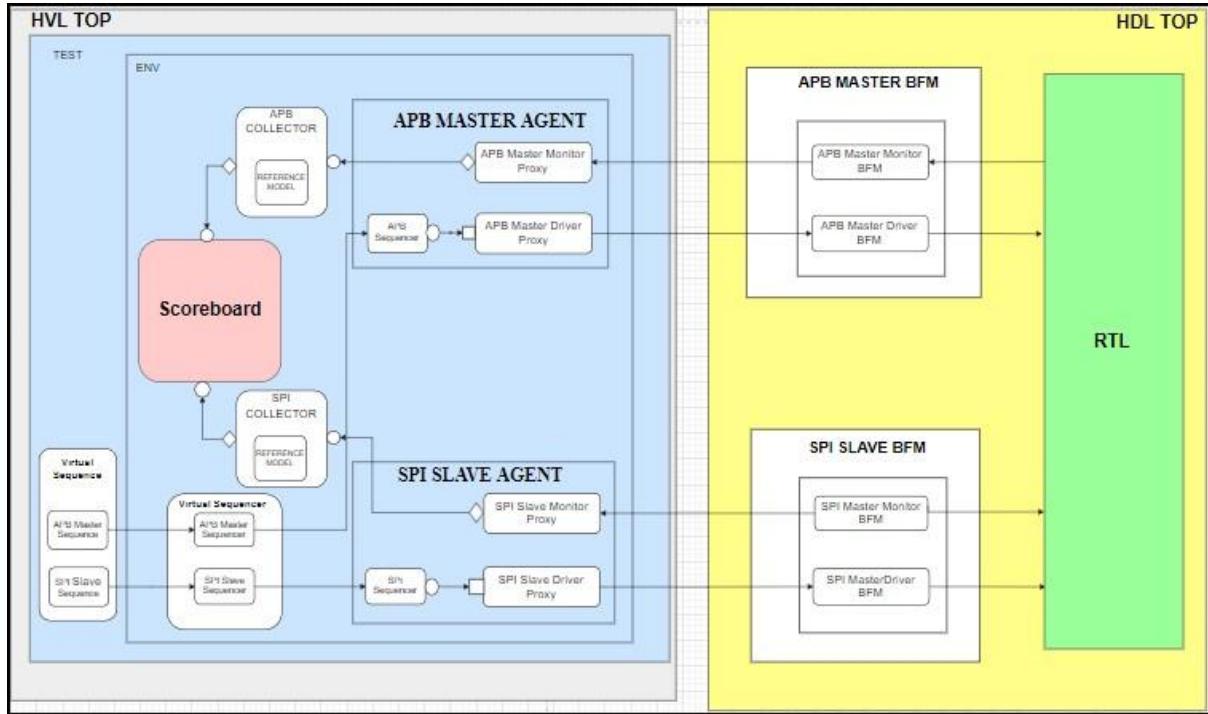


Fig 2.1 pulpino_spi_master_avip Architecture

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 APB I/F through the BFM Proxy and BFM and gets the data from monitor BFM and uses the data to do checks using scoreboard and coverage.

HDL TOP consists of the design part which is timed and synthesizable, Clock and reset signals are generated in the HDL TOP. Bus Functional Models (BFMs) i.e synthesizable part of drivers and monitors are present in HDL TOP, BFMs also have the back pointers to its 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_ip_verification](#)

git clone

[git@github.com:mirafra-software-technologies/pulpino_spi_master_ip_verification.git](#)

3. After cloning, change the directory to the cloned repository

cd pulpino_spi_master_ip_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_ip_verification repository

git fetch

7. Check all branches present in the pulpino_spi_master_ip_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_ip_verification [pulpino_spi_master_ip_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_ip_verification/sim/questasim

Note: To Compile, simulate, regression and for coverage you must be in the specified path i.e, *pulpino_spi_master_ip_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 `pulpino_spi_master_ip_verification_compile.log` to view the compiled files

gvim pulpino_spi_master_ip_verification_compile.log

3.2.2 Simulation

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

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

Example:

- To view the log file

gvim <test_name>/<test_name>.log

Ex:

*gvim
pulpino_spi_master_ip_basic_write_test/pulpino_spi_master_ip_basic_write_test.log*

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

- To view waveform

vsim -view <test_name>/waveform.wlf &

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

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

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

The screenshot shows the Questasim WLF (Waveform) window. The menu bar includes File, Edit, View, Compile, Simulate, Add, Structure, Tools, Layout, Bookmarks, Window, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The status bar at the bottom shows 'File: waveform - Default'.

The main area displays a table with columns: Instance, Design unit, Design unit type, Top Category, Visibility, and Total cove. The table lists numerous design instances under the root 'hdl_top'. Some instances are expanded to show their internal structure, such as 'apb_master_agent_bfm_h' which contains 'intf', 'apb_master_drv_bfm_h', 'drive_to_bfm', 'drive_setup_state', 'waiting_in_access_state', 'detect_wait_state', 'apb_master_mon_bfm_h', 'sample_data', 'spi_intf', and 'DUT'. Other instances include 'spi_agent_bfm_h', 'uvm_pkg', and 'questa_uvm_pkg'. The 'Visibility' column shows values like '+acc=<...'. The 'Total cove' column is mostly blank or shows '-'.

Instance	Design unit	Design unit type	Top Category	Visibility	Total cove
hdl_top	hdl_top	Module	DU Instance	+acc=<...	-
apb_intf	apb_if	Interface	DU Instance	+acc=<...	-
apb_master_agent_bfm_h	apb_master...	Module	DU Instance	+acc=<...	-
intf	apb_if	Interface	DU Instance	+acc=<...	-
apb_master_drv_bfm_h	apb_master...	Interface	DU Instance	+acc=<...	-
drive_to_bfm	apb_master...	Task	-	+acc=<...	-
drive_setup_state	apb_master...	Task	-	+acc=<...	-
waiting_in_access_state	apb_master...	Task	-	+acc=<...	-
detect_wait_state	apb_master...	Task	-	+acc=<...	-
apb_master_mon_bfm_h	apb_master...	Interface	DU Instance	+acc=<...	-
sample_data	apb_master...	Task	-	+acc=<...	-
spi_intf	spi_if	Interface	DU Instance	+acc=<...	-
DUT	apb_spi_ma...	Module	DU Instance	+acc=<...	-
spi_agent_bfm_h	spi_slave_a...	Module	DU Instance	+acc=<...	-
intf	spi_if	Interface	DU Instance	+acc=<...	-
spi_slave_drv_bfm_h	spi_slave_d...	Interface	DU Instance	+acc=<...	-
spi_slave_mon_bfm_h	spi_slave_m...	Interface	DU Instance	+acc=<...	-
uvm_pkg	uvm_pkg	VIPackage	Package	+acc=<...	-
questa_uvm_pkg	questa_uv...	VIPackage	Package	+acc=<...	-

Fig 3.2 Questasim WLF window

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

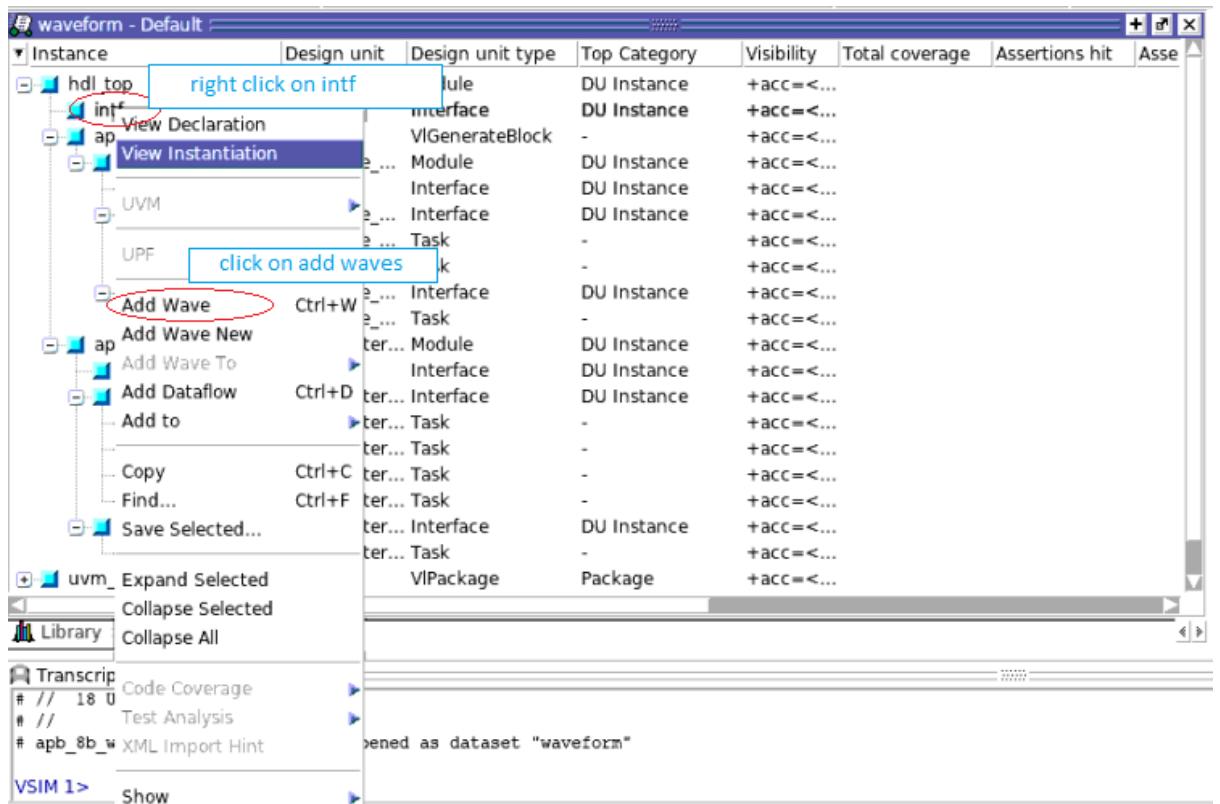


Fig 3.2 Screenshot of adding waves in wave window

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

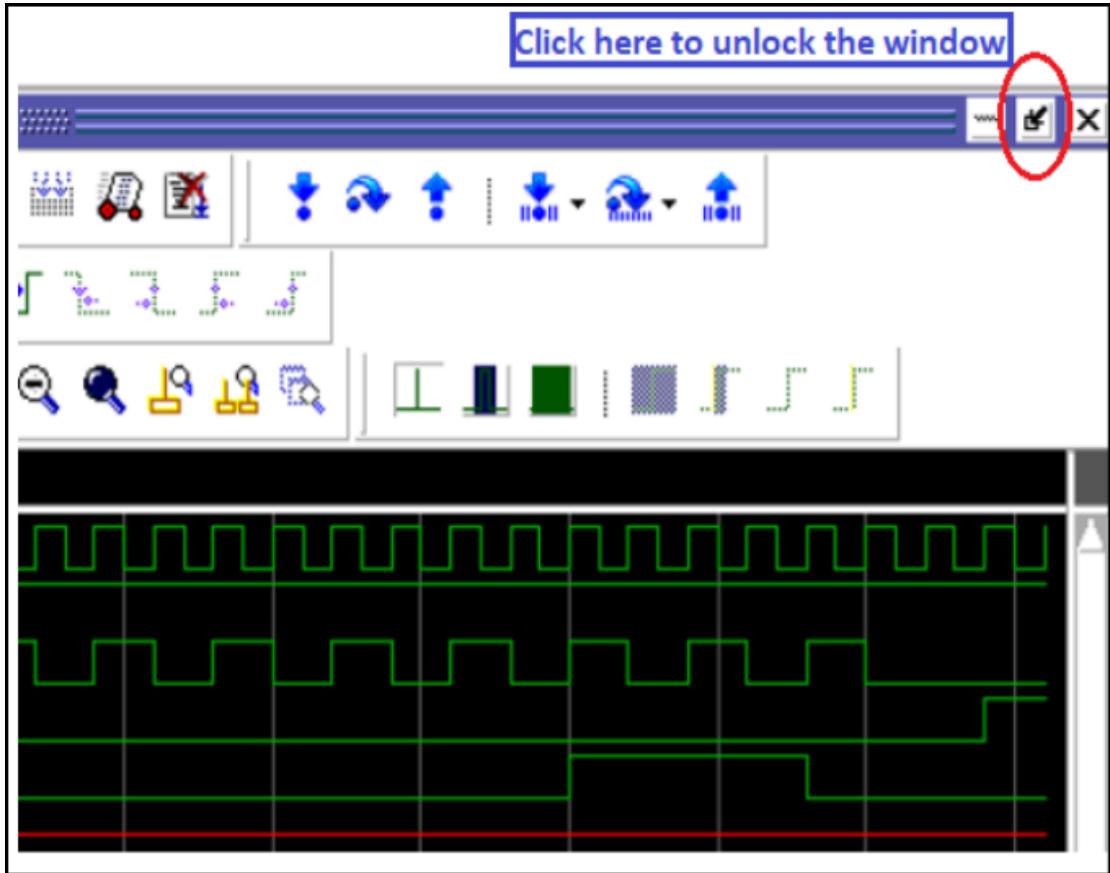


Fig 3.4 Screenshot of unlocking the wave window

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

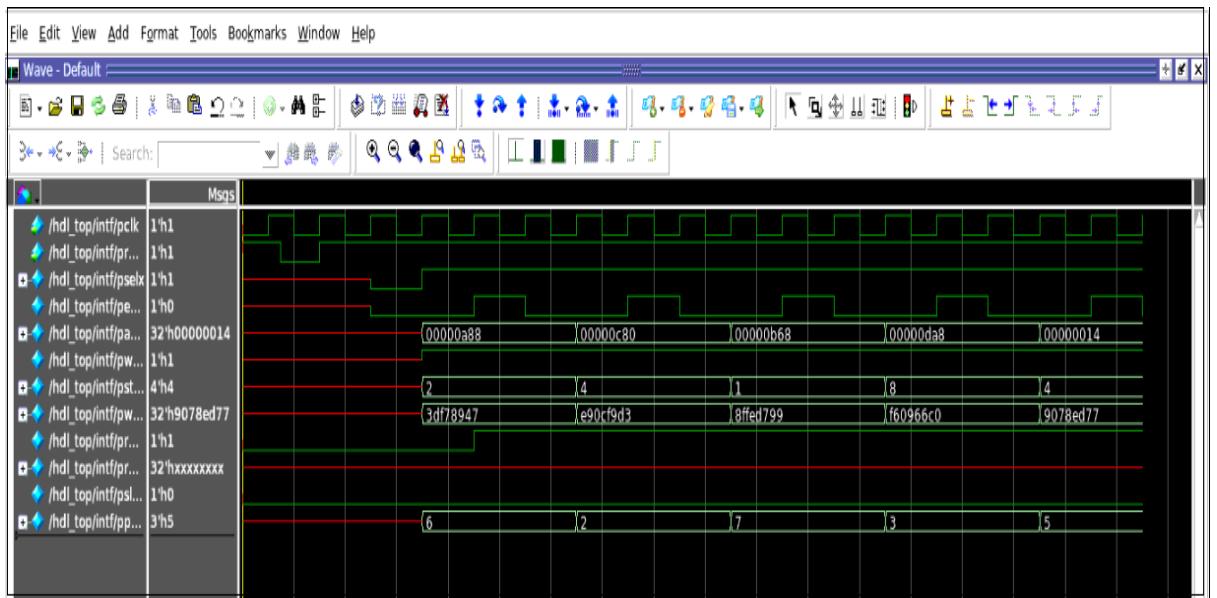


Fig 3.5 Screenshot of unlocked wave window with signals

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

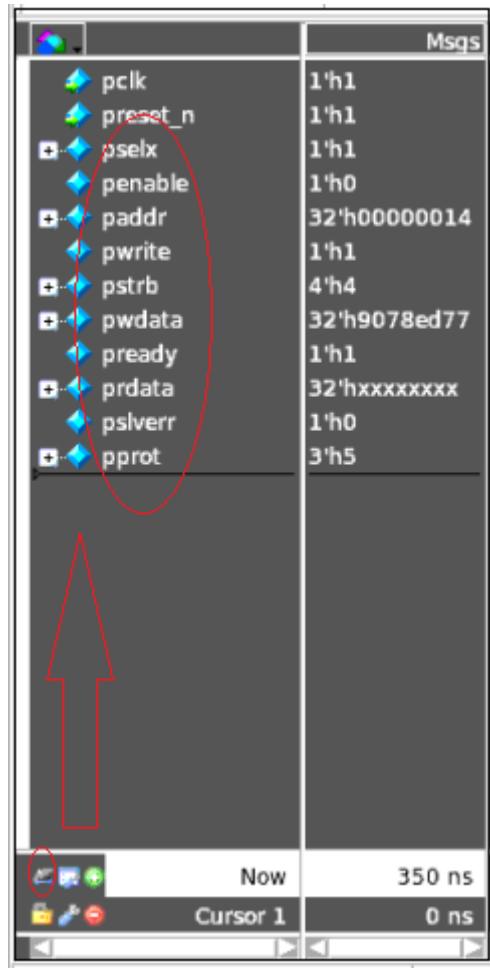


Fig 3.6 Screenshot showing way to see the name of signals

- For the analysis of waveform, go through the link below

[Waveform Viewer](#)

3.2.3 Regression

- To run regression for all test case

make regression testlist_name=<regression_testlist_name.list>

Ex: *make regression testlist_name=pulpino_spi_master_ip_verification_regression.list*

Note: You can find all the test case names in the path given below

pulpino_spi_master_ip_verification/src/hvl_top/testlists/pulpino_spi_master_ip_verification_regression.list

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

ls

```
pulpino_spi_master_ip_basic_read_test_18032022-152218
pulpino_spi_master_ip_basic_write_read_reg_test_18032022-152334
pulpino_spi_master_ip_basic_write_reg_test_18032022-152339
pulpino_spi_master_ip_basic_write_test_18032022-152229
pulpino_spi_master_ip_rand_reg_test_18032022-152343
pulpino_spi_master_ip_rand_test_18032022-152234
```

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 *pulpino_spi_master_ip_basic_write_read_reg_test*

Go inside the directory of interested testcase with the date

pulpino_spi_master_ip_basic_write_read_reg_test_18032022-152334

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

pulpino_spi_master_ip_basic_write_read_reg_test.log

Path:

pulpino_spi_master_ip_basic_write_read_reg_test_18032022-152334/pulpino_spi_master_ip_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 pulpino_spi_master_ip_basic_write_read_reg_test/html_cov_report/index.html &

Ex:

firefox pulpino_spi_master_ip_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 *pulpino_spi_master_ip_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

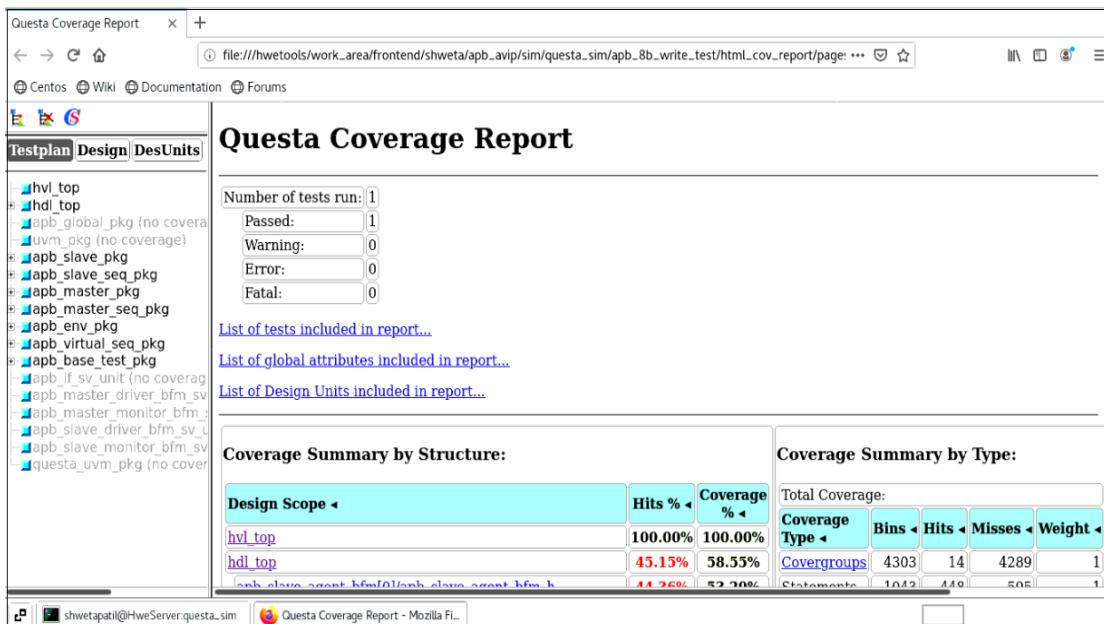


Fig 3.8 Coverage Report

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

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

3.9 Screenshot of opening covergroups

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

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

3.10 Screenshot of opening slave covergroup

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

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

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

Scope: [/apb_slave_pkg/apb_slave_coverage](#)

Covergroup type:

apb_slave_covergroup

Summary	Total Bins	Hits	Hit %	Search: <input type="text"/>			
Coverpoints	102	5	4.90%				
Crosses	2048	1	0.04%				
CoverPoints	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %	
PADDR_CP	32	1	31	3.12%	3.12%	3.12	
PRDATA_CP	32	1	31	3.12%	3.12%	3.12	
PSELX_CP	2	1	1	50.00%	50.00%	50.00	
PSLVERR_CP	2	1	1	50.00%	50.00%	50.00	
PWDATA_CP	32	0	32	0.00%	0.00%	0.00	
PWRITE_CP	2	1	1	50.00%	50.00%	50.00	

Fig 3.12 Screenshot of Coverpoints 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

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_ip_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 [apb_compile.log](#) to view the compiled files
vim pulpino_spi_master_ip_verification_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_ip_verification_compile/src/hvl_top/testlists/pulpino_spi_master_ip_verification_regression.list

2. To view the log file

gvim <test_name>/<test_name>.log

Ex:

gvim

pulpino_spi_master_ip_basic_write_test/pulpino_spi_master_ip_basic_write_test.log

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

```
----- 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

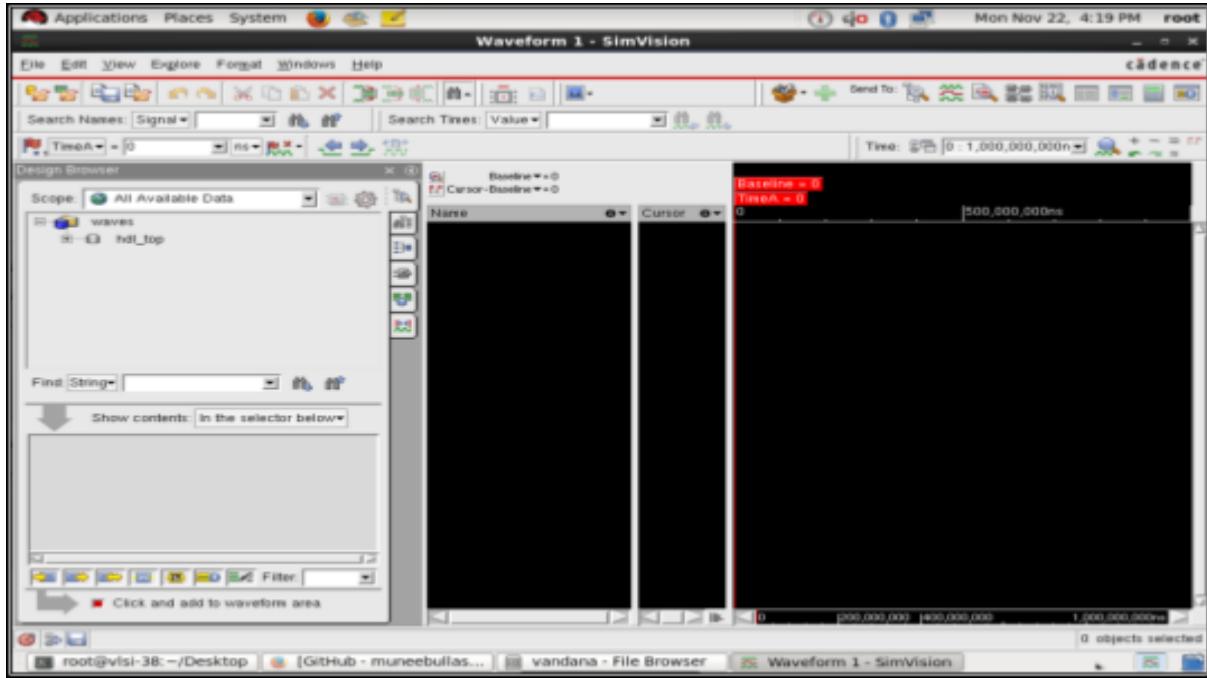


Fig 3.16 Cadence waveform window

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

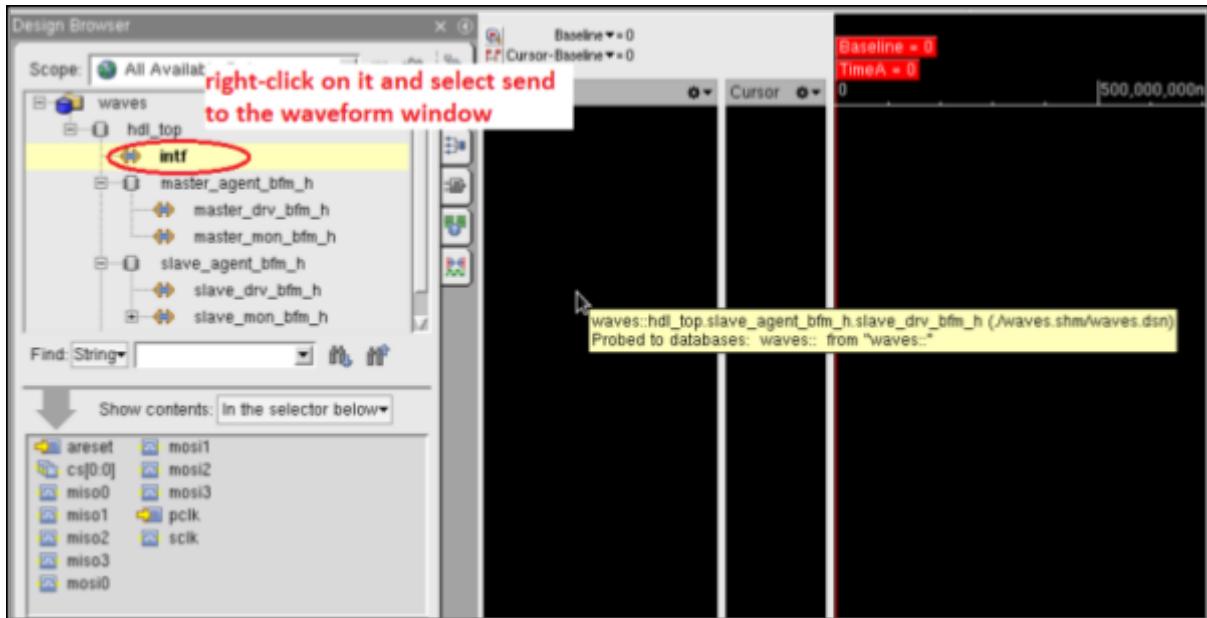


Fig 3.17

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

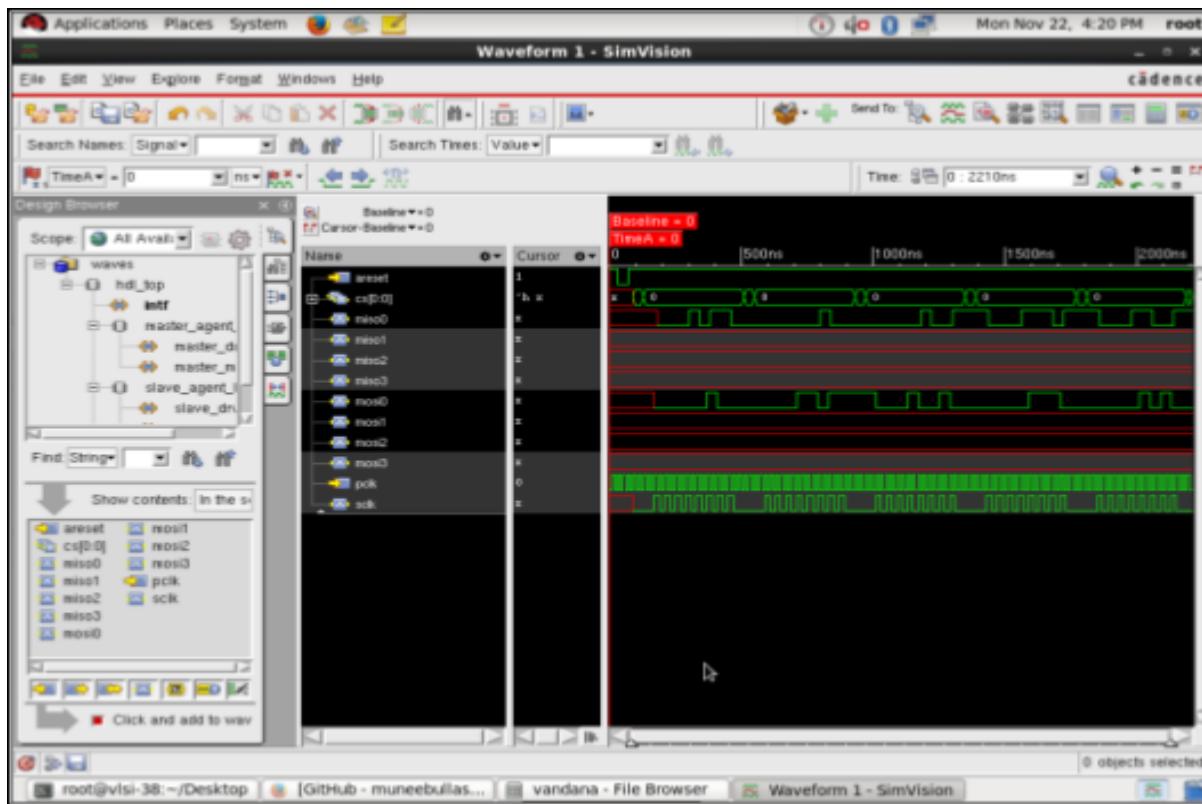


Fig 3.18

3.3.3 Coverage

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

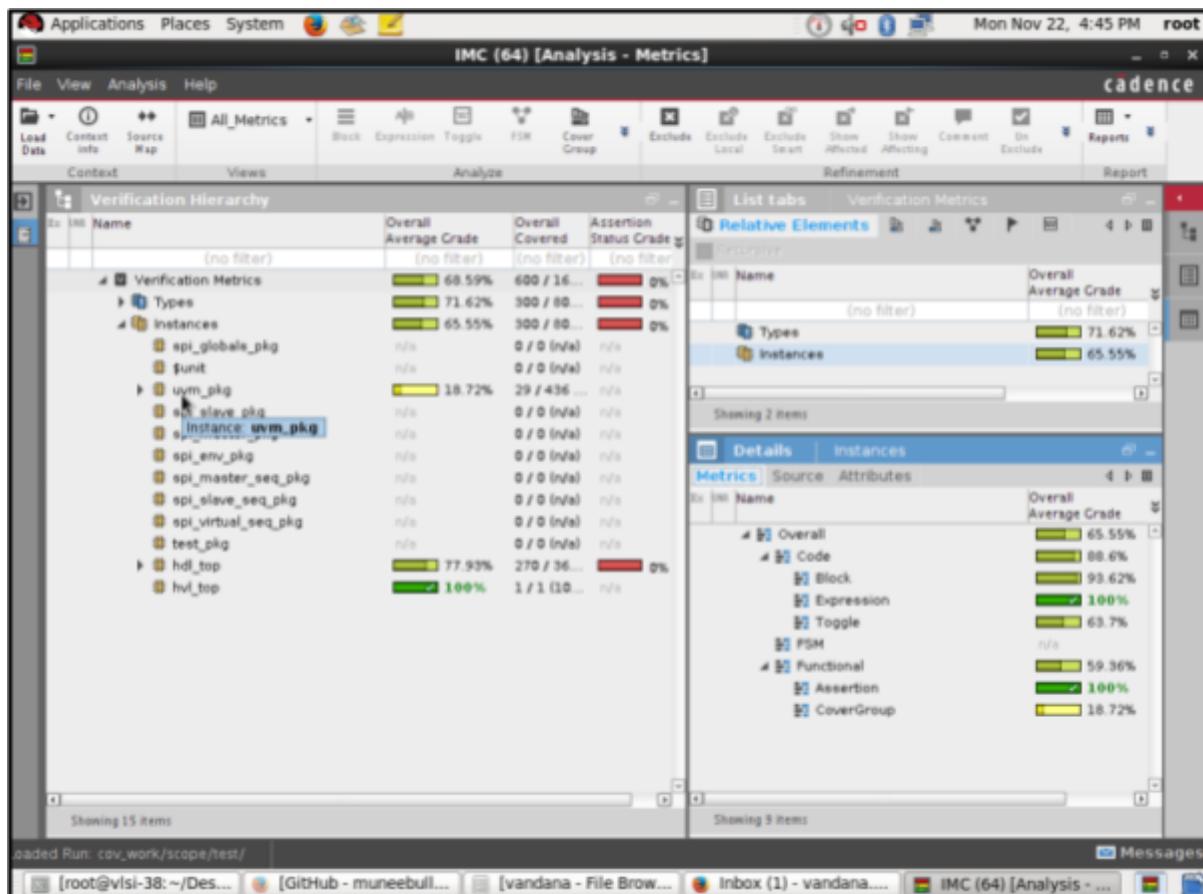


Fig 3.19

- 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

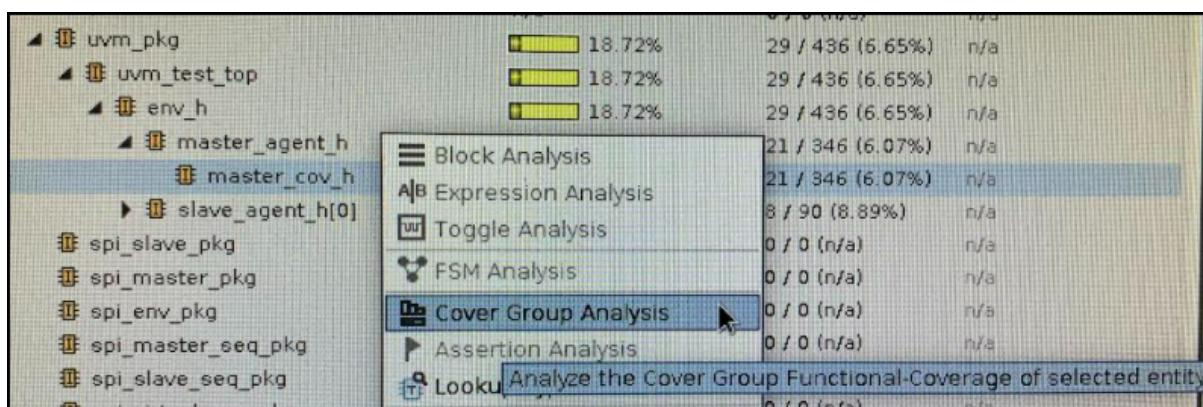
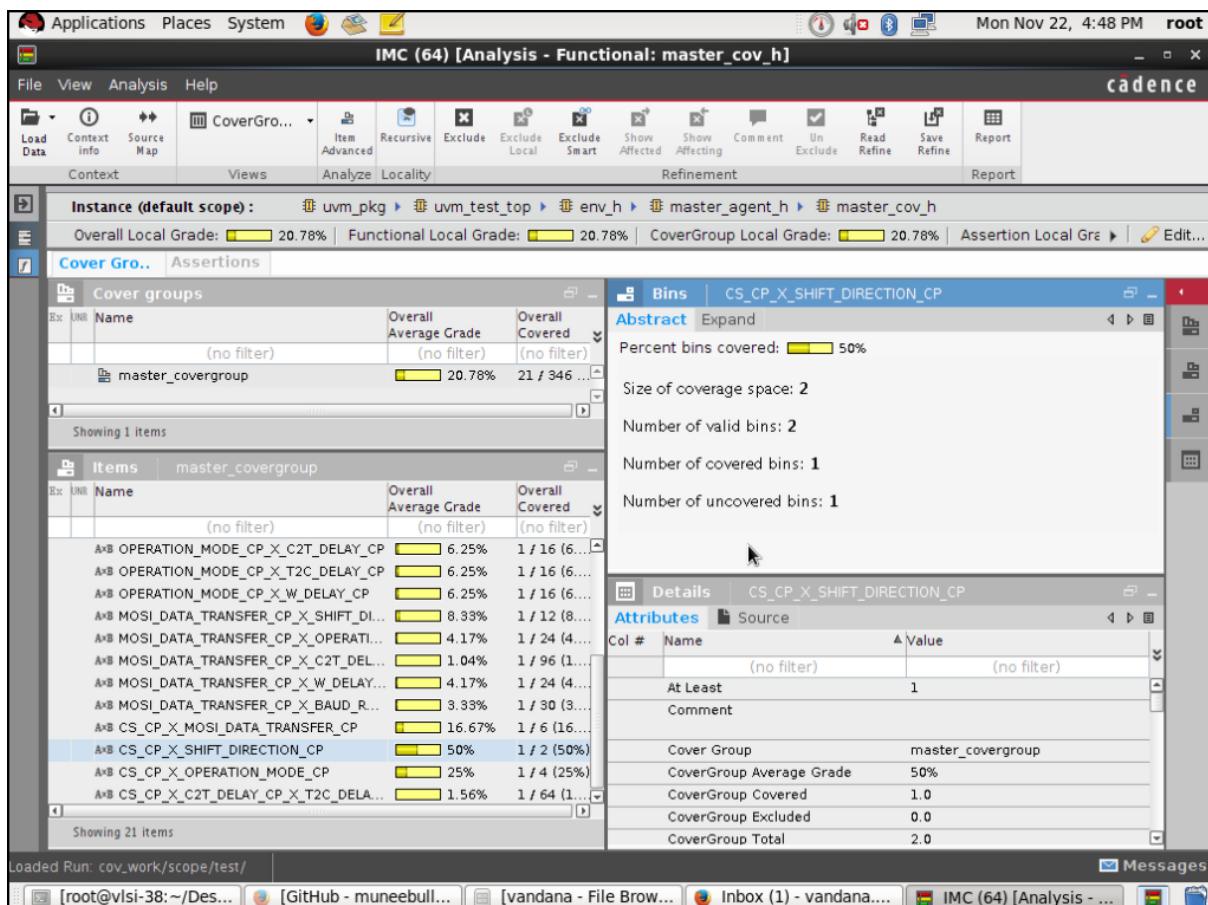
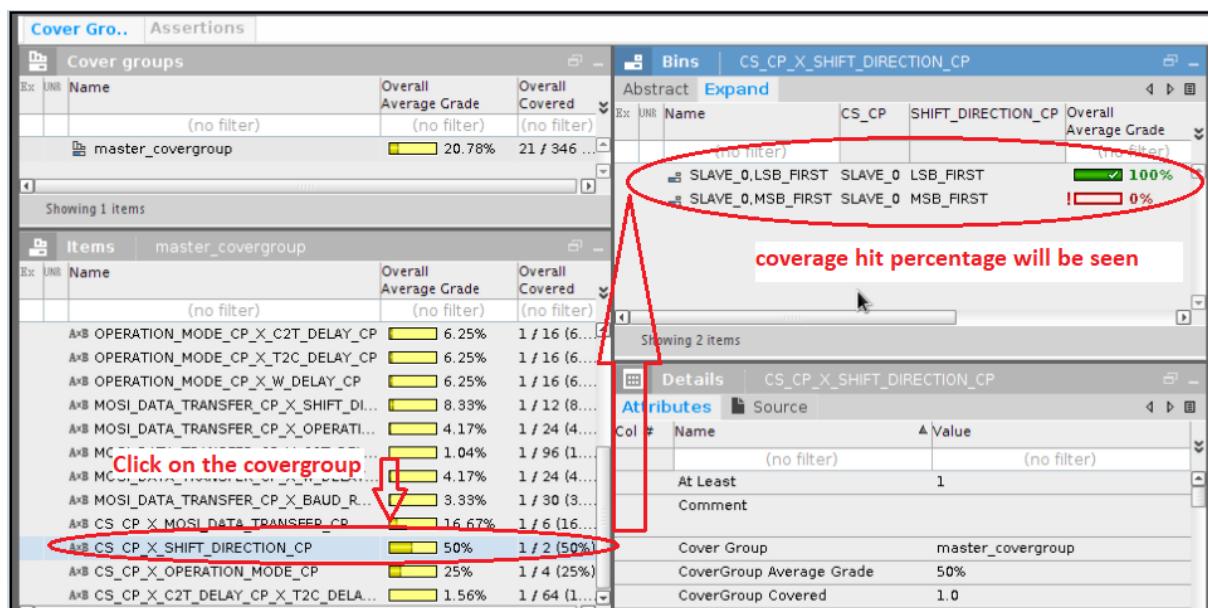


Fig 3.20

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



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

4.1 APB Debugging Flow

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

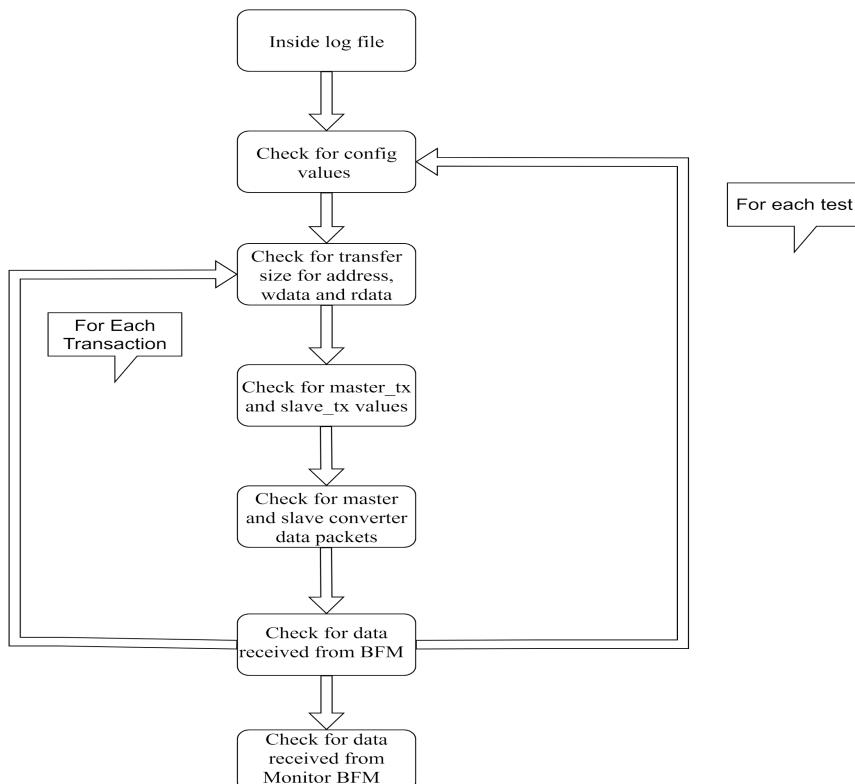


Fig 4.1 Debugging flow

4.1.1 Check for Configuration Values

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

For more information on Configurations please visit the following link:

[Configuration Doc](#)

4.1.1(a) Master agent configurations

Master agent configurations Includes

Table 4.1 master configurations

Configurations	conditions for the configurations
No of Slaves	which should not equal to zero
has_coverage	Which indicates the coverage connection

```
UVM_INFO .../src/dv/hvl_top/test//pulpino_spi_master_ip_base_test.sv(80) @ 0: uvm_test_top [pulpino_spi_master_ip_basic_write
est]
apb_master_agent_CONFIG
-----
Name          Type           Size  Value
-----
apb_master_agent_config apb_master_agent_config -    @497
  is_active      integral        1    1
  has_coverage   integral        1    1
  no_of_slaves   integral       32   'd1
```

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

```
UVM_INFO ../../src/dv/hvl_top/test/pulpino_spi_master_ip_base_test.sv(170) @ 0: uvm_test_top [pulpino_spi_master_ip_basic_write
test]
pulpino_spi_master_ip_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

```
UVM_INFO ../../src/dv/hvl_top/test/pulpino_spi_master_ip_base_test.sv(71) @ 0: uvm_test_top [pulpino_spi_master_ip_basic_write
est]
pulpino_spi_master_ip_ENV_CONFIG
-----
Name          Type           Size  Value
-----
pulpino_spi_master_ip_env_cfg_h pulpino_spi_master_ip_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

APB follows one specific rule related to the transfer size of pwnode and prdata.

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

```
# 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 of pwnode or prdata.

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.

```
UVM_INFO ../../src/dv/hvl_top/test/sequences/master_sequences//apb_master_basic_write_seq.sv(58) @ 2560: uvm_test_top.pulpino_s  
i_master_ip_env.apb_master_agent.apb_master_seqr_h@apb_master_basic_seq_h [CLOCK_DIV_REG_SEQ] clkddiv_reg_seq =  
-----  
Name          Type      Size  Value  
-----  
req           apb_master_tx -    @1428  
pselx         string    7    SLAVE_0  
paddr         integral   32   'h1a102004  
pwrite        string    5    WRITE  
pwdata        integral   32   'h0  
transfer_size string    6    BIT_32  
pstrb         integral   4    'b1111  
pprot         string    25   NORMAL_SECURE_INSTRUCTION  
prdata        integral   32   'h0  
pslverr       string    8    NO_ERROR  
no_of_wait_states_detected integral 32   'd0
```

Fig 4.6 master_tx values

req	spi_slave_tx	-	@1411
begin_time	time	64	540
depth	int	32	'd2
parent sequence (name)	string	24	spi_fd_basic_slave_seq_h
parent sequence (full name)	string	101	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
spi_fd_basic_slave_seq_h	sequencer	76	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
master_in_slave_out[0]	integral	8	'hed
master_in_slave_out[1]	integral	8	'h84
master_in_slave_out[2]	integral	8	'hbc
master_in_slave_out[3]	integral	8	'hdb

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/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(41) @ 2560: reporter [apb_master_seq_item_conv
class] -----
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(45) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize pprot = 001
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(49) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize pselx = 1
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(52) @ 2560: reporter [apb_master_seq_item_conv
class] After randomizing pwrite = 1
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(55) @ 2560: reporter [apb_master_seq_item_conv
class] after writing addr = 1a102004
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(58) @ 2560: reporter [apb_master_seq_item_conv
class] after randomizing pwdata = 0
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(61) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize pstrb = 1111
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(65) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize pslverr = 0
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(68) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize prdata = 0
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(71) @ 2560: reporter [apb_master_seq_item_conv
class] After randomize no_of_wait_states = 0
: UVM_INFO ../../src/dv/hvl_top/apb_master_agent//apb_master_seq_item_converter.sv(73) @ 2560: reporter [apb_master_seq_item_conv
class] -----
```

Fig 4.8 converted data of master req

UVM_INFO .../src/dv/hvl_top/test/sequences/slave_sequences//spi_fd_basic_slave_seq.sv(44) @ 540: uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h@spi_fd_basic_slave_seq_h [spi_fd_basic_slave_seq] slave_seq =			
Name	Type	Size	Value
req	spi_slave_tx	-	@1411
begin_time	time	64	540
depth	int	32	'd2
parent sequence (name)	string	24	spi_fd_basic_slave_seq_h
parent sequence (full name)	string	101	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
.spi_fd_basic_slave_seq_h	sequencer	76	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
master_in_slave_out[0]	integral	8	'hed
master_in_slave_out[1]	integral	8	'h84

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 BFM s. The master driver BFM will drive paddr,pwrite,pstrobe,pwdtdata signals and samples the prdata,pslverr,pready depending on configurations of master and similarly slave driver BFM will drive the prdata,pready,pslverr signal and samples the paddr,psel,pwrite,pwdtdata 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/apb_master_agent.bfm/apb_master_driver.bfm.sv(83) @ 50: reporter [APB_MASTER_DRIVER_BFM] data_pac  
ket= [mvsit01 psolverr:0 paddr:2 psclx:1 psrb:15 prdata:0 paddrs:437264388 prdata:0 ps_of:wait states:0]
```

Fig 4.10 master.bfm_struct data

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/dv/hvl_top/apb_master_agent//apb_master_driver_proxy.sv(109) @ 2560: uvm_test_top.pulpino_spi_master_ip_env.apb_master_agent.apb_master_drv_proxy_h [apb_master_driver_proxy] REQ-MASTER_TX			
Name	Type	Size	Value
req	apb_master_tx	-	@1428
pselx	string	7	SLAVE_0
paddr	integral	32	'h1a102004
pwrite	string	5	WRITE
pwdata	integral	32	'h0
transfer_size	string	6	BIT_32
pstrb	integral	4	'b1111
pprot	string	25	NORMAL_SECURE_INSTRUCTION
prdata	integral	32	'h0
pslverr	string	8	NO_ERROR
no_of_wait_states_detected	integral	32	'd0

Fig 4.13 master_driver_bfm values

UVM_INFO ../../src/dv/hvl_top/spi_slave_agent//spi_slave_driver_proxy.sv(129) @ 540: uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_drv_proxy_h [spi_slave_driver_proxy] Received packet from spi_slave sequencer : ,			
Name	Type	Size	Value
req	spi_slave_tx	-	@1411
begin_time	time	64	540
depth	int	32	'd2
parent sequence (name)	string	24	spi_fd_basic_slave_seq_h
parent sequence (full name)	string	101	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
.spi_fd_basic_slave_seq_h	sequencer	76	uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_seqr_h
master_in_slave_out[0]	integral	8	'hed
master_in_slave_out[1]	integral	8	'h84

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/dv/hvl_top/apb_master_agent//apb_master_monitor_proxy.sv(98) @ 2660: uvm_test_top.pulpino_spi_master_ip_env.apb_master_agent.apb_master_mon_proxy_h [apb_master_monitor_proxy] Received packet from master monitor bfm: ,				
Name	Type	Size	Value	
master_packet	apb_master_tx	-	@1256	
pselx	string	7	SLAVE_0	
paddr	integral	32	'h1a102008	
pwrite	string	5	WRITE	
pwdata	integral	32	'hffffffff	
transfer_size	string	0	"	
pstrb	integral	4	'b1111	
pprot	string	29	PRIVILEGED_SECURE_INSTRUCTION	
prdata	integral	32	'h0	
pslverr	string	8	NO_ERROR	
no_of_wait_states_detected	integral	32	'd0	

Fig 4.15 master_monitor values

UVM_INFO ../../src/dv/hvl_top/spi_slave_agent//spi_slave_monitor_proxy.sv(260) @ 2560: uvm_test_top.pulpino_spi_master_ip_env.spi_slave_agent_h[0].spi_slave_mon_proxy_h [spi_slave_monitor_proxy] Received packet from spi_slave MONITOR BFM : ,				
Name	Type	Size	Value	
spi_slave_tx	spi_slave_tx	-	@1419	
master_in_slave_out[0]	integral	8	'hed	
master_in_slave_out[1]	integral	8	'h84	
master_in_slave_out[2]	integral	8	'hbc	
master_in_slave_out[3]	integral	8	'hdb	
master_in_slave_out[4]	integral	8	'h0	
master_in_slave_out[5]	integral	8	'h0	
master_out_slave_in[0]	integral	8	'hoff	
master_out_slave_in[1]	integral	8	'hff	
master_out_slave_in[2]	integral	8	'hf	
master_out_slave_in[3]	integral	8	'ha9	
master_out_slave_in[4]	integral	8	'h0	
master_out_slave_in[5]	integral	8	'h0	

Fig 4.16 slave_monitor values

4.2 Scoreboard Checks

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

SCOREBOARD COMPARISONS-----				
UVM_INFO ../../src/dv/hvl_top/env//pulpino_spi_master_ip_scoreboard.sv(128) @ 5060: uvm_test_top.pulpino_spi_master_ip_env.pulpino_spi_master_ip_scoreboard_h [pulpino_spi_master_ip_scoreboard] apb_pwdata from apb_master and master_out_slave_in from slave is not equal				
UVM_INFO ../../src/dv/hvl_top/env//pulpino_spi_master_ip_scoreboard.sv(129) @ 5060: uvm_test_top.pulpino_spi_master_ip_env.pulpino_spi_master_ip_scoreboard_h [SB_PWDATA_MATCHED WITH MOSI0] Master APB_DATA = 'h0 and Slave SPI_DATA = 'hffff0fa90000				
UVM_INFO ../../src/dv/hvl_top/env//pulpino_spi_master_ip_scoreboard.sv(133) @ 5060: uvm_test_top.pulpino_spi_master_ip_env.pulpino_spi_master_ip_scoreboard_h [pulpino_spi_master_ip_scoreboard] --				

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_slave_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:						
	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
① /spi_master_uvm_pkg/spi_master_apb_if_status/cg_vals	26	6	20	23.07%	42.70%	42.70%
① /spi_master_uvm_pkg/spi_master_apb_if_clkdiv/cg_vals	64	1	63	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_spicmd/cg_vals	64	1	63	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_spiaidr/cg_vals	64	1	63	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_spilen/cg_vals	192	3	189	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_spidum/cg_vals	128	2	126	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_txfifo/cg_vals	64	1	63	1.56%	1.56%	1.56%
① /spi_master_uvm_pkg/spi_master_apb_if_rxfifo/cg_vals	64	0	64	0.00%	0.00%	0.00%
① /spi_master_uvm_pkg/spi_master_apb_if_intcfg/cg_vals	132	6	126	4.54%	18.75%	18.75%
① /spi_master_uvm_pkg/spi_master_apb_if_coverage/ra_cov	28	17	11	60.71%	61.98%	61.98%
① /spi_slave_pkg/spi_slave_coverage/spi_slave_covergroup	90	2	88	2.22%	9.37%	9.37%
① /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

Covergroups/Instances	Total Bins	Hits	Misses	Hits %	Goal %	Coverage %
① Covergroup apb_master_covergroup	145	14	131	9.65%	29.34%	29.34%
① Instance Vapb_master_pkg::apb_master_c...verage::apb_master_covergroup	145	14	131	9.65%	29.34%	29.34%

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 %
PADDR_CP	2	0	2	0.00%	0.00%	0.00%
PPROT_CP	8	6	2	75.00%	75.00%	75.00%
PRDATA_CP	3	1	2	33.33%	33.33%	33.33%
PSEL_CP	1	1	0	100.00%	100.00%	100.00%
PSLVERR_CP	2	1	1	50.00%	50.00%	50.00%
PSTRB_CP	16	1	15	6.25%	6.25%	6.25%
PWDATA_CP	3	1	2	33.33%	33.33%	33.33%
PWRITE_CP	2	1	1	50.00%	50.00%	50.00%

Fig 4.21 master_coverage coverpoint

Figure 4.21 shows all the coverpoints included in master coverage

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
PADDR_CP_X_PRDATA_CP	6	0	6	0.00%	0.00%	0.00%
PADDR_CP_X_PWDATA_CP	6	0	6	0.00%	0.00%	0.00%
PSTRB_CP_X_PRDATA_CP	48	1	47	2.08%	2.08%	2.08%
PSTRB_CP_X_PWDATA_CP	48	1	47	2.08%	2.08%	2.08%

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 %
① MISO_DATA_TRANSFER_CP	6	0	6	0.00%	0.00%	0.00%
① MOSI_DATA_TRANSFER_CP	6	0	6	0.00%	0.00%	0.00%
① OPERATION_MODE_CP	4	1	3	25.00%	25.00%	25.00%
① SHIFT_DIRECTION_CP	2	1	1	50.00%	50.00%	50.00%

Fig 4.23 slave_coverage coverpoint

Figure 4.23 shows all the coverpoints included in slave coverage

Crosses	Total Bins	Hits	Misses	Hit %	Goal %	Coverage %
① OPERATION_MODE_X_MASTER_IN_slave_OUT	24	0	24	0.00%	0.00%	0.00%
① OPERATION_MODE_X_MASTER_OUT_slave_IN	24	0	24	0.00%	0.00%	0.00%
① SHIFT_DIRECTION_X_MASTER_IN_slave_OUT	12	0	12	0.00%	0.00%	0.00%
① 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 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 occurring.

```

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

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(pcik), after every 10ns it will be toggled. Once the pcik 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 pcik the cs and sclk should be in idle state.

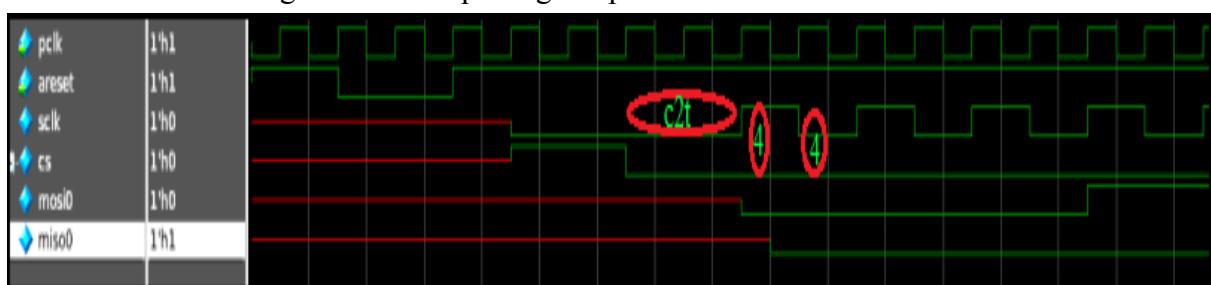


Fig 4.31 wave form for 32 bit full duplex test with c2t delay

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.

- 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.

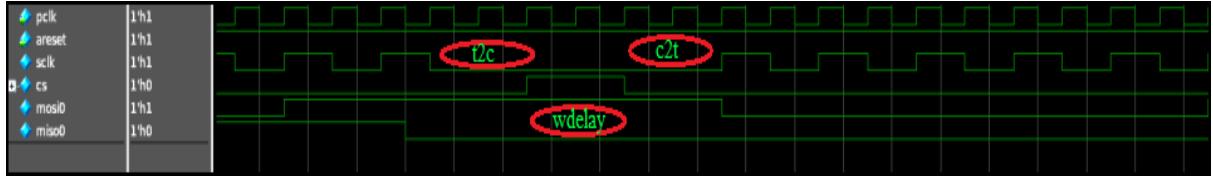


Fig 4.32 wave form for 32 bit full duplex test with all the delays

- 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.

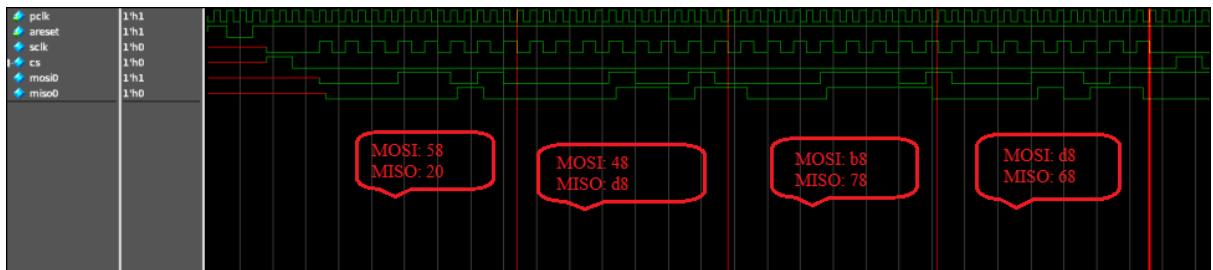


Fig 4.33 wave form for 32bit data transfer(1st transaction)

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>