SIP CONTROLLER FOR MASTER CORE VERIFICATION USING UVM

[1]ShyamalaS.C, [2]Kalpana.S, [3]Manasa.B, [4]Bindu.L

Assistant Professor
ECE Department PESITM, Shivamogga.karnataka, India

1. INTRODUCTION

This section will describe the features of SPI (Serial Peripheral Interface) protocol using UVM (Universal Verification Methodology) and ensuring its correct working functionality.

SPI Master Core is a synchronous serial interface which is widely used to provide economical board-level interface between different devices such as microcontrollers, ADCs, DACs and other peripherals, although there is no fix standards for a serial transfers but there are two most popular protocols are there which has been accepted by industry.

A serial data link standard named Serial Peripheral Interface is verified using UVM. This protocol demonstrates the ability to transform incoming parallel communication from a wishbone bus, into serial communication that is being transferred using the SPI protocol. Serial Peripheral Interface (SPI) is Serial synchronous interface that facilitates the transfer of synchronous serial data in full duplex mode. SPI use for communication with peripheral devices where we want to transfer data very fast and within real time constraints. SPI master core consist of three parts Serial interface, Clock generation and Wishbone interface.

The SPI core has seven 32-bit register through the wishbone compatible interface. The serial interface consists of slave select lines, serial clock lines, as well as input and output data lines. It communicates in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select line. Serial Peripheral Interface of symmetrical structure can be simulated using Questa 10.0b. It is a popular interface used for connecting peripherals to each other and to microprocessors.

In This thesis an implemented code coverage which gives an idea about how much percentage of given design has been triggered with written verification plan, In this thesis functional coverage has also done which can cover every possible working functionality of SPI core design and with the use of assertion can track some of the RTL level bug in available design. In this thesis twelve number of test cases has used which covers 100% functional coverage and 100% code coverage and got two RTL level bugs which helps design engineer to improve with the design coding style.

Block Diagram

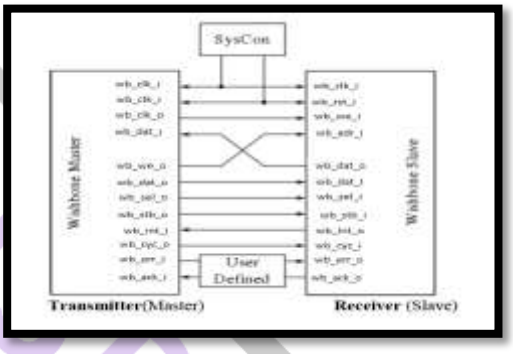


Figure 1 Signal architecture and bus description.

UVM OBJECT/COMPONENT DESCRIPTION

Transaction Class:

It is a class where we define all the signals which are needed for transactions. All the input signals are defined as a "rand" data type where output signals are defined as a "bit" data type .

- During verification of this project we have some handshaking signals and some signals which should act as input to master and output to salve or vice-versa.
- So to achieve this we can make one or two transaction classes. But in this verification, two transaction classes i.e., one for master and other for slave are considered. This class is extended from uvm_sequence_item as it has to be randomized and passed everywhere in the test bench.
- This class is then registered to factory using macro `uvm_object_utils. After this, declared all the signals required for master – slave transaction. The signal that has to be randomized is declared as “rand” variables. "rand " is a data type in system Verilog which is use to randomize the variable available inside the class.
- All the signals are registered to factory using respective macros. Here, defined some constraints according to the project in “master transaction”. Constraint is a data type in system Verilog which allows the randomize data to be randomize within a given constraint so that we can have a control over a randomization which is a advanced feature in

system Verilog then Verilog where we didn't having this facility .These are as follows:-

- Change byte select signal wb_sel_i according to the number of bytes to be selected, here considered as 4'b1111. These constraints will randomize the data as required in the protocol.

Virtual interface

Interface is defined as a bundle of signals where all the input signals are defined as a "logic" data type and outputs are defined as a "wire" data type.

- Interface class comes with class name "interface".
- In this class will pass input signal as a clock with a "bit" data type.
- Inside an interface class will be having clocking block for master monitor, master driver, slave monitor and slave _driver .
- Clocking blocks are nothing but its a block where will give direction to our signal according to the protocol.
- Inside a clocking block will be also taking a input and output skew in order to avoid meta stability issue. Here default value of skew is input #1 output #0.
- Inside a sensitivity list of clocking block will put an event clock based on which signals should respond.
- In order to activate a given directions inside a clocking block will be taking "modport" which is again a data type which enables the given direction for each signals inside a clocking block by passing an argument as a clocking block name .
- In this interface assertions can also be added which helps the verification engineer to track the register level bugs very easily.
- Interface plays a very important character in this verification environment.
- There are four clocking blocks are there , Hence master driver clocking block will be going to connect with the master driver , slave driver clocking block will be going to connect with the slave_driver , master monitor clocking block will be going to connect with the master monitor and slave monitor clocking block will be going to connect with the slave monitor.
- Interface are the static component which are non-synthesizable
- If will remove the clocking block then it will be synthesizable
- This is an advantage in UVM over Verilog that will be having an interface concept which reduces complexity and increase readability.

Master_agent_top

It is class which enclosed the master_agent class properties as well as methods.

- Master_agent_top is extended from uvm_envireonment.
- With the `uvm_component_utils the factory registration of master_agent_top has been done.
- Uvm_component_utils is a macros in UVM which is used for factory registration.
- With the **function_new** will be able create a class having a name master_agent_top and it is overridden to the parent class by super. New function.
- In the **build__phase** of master_agent_top will create master_agent, for that will allocate a handle of master_agent.
- In the **run__phase** will be printing a topology with uvm_top.print_topology () macro name.

Master_agent

It is a class which enclosed the configuration class , driver class , sequencer class and monitor class properties as well as methods.

- master agent is extended from uvm_agent .

Factory registration of this class will be done using UVM macros `uvm_component_utils by passing master_agent as an argument.

- **function new** is use to create a class having a name called master_agent which is gets overridden on the parent class.
- In the **build phase** of master agent will get the configuration data from uvm_config_db which is a macro stores the information about config class , Hence if will get the config_db then will create a master_mmonitor else will be displaying an error massage using macro `uvm_error.
- In the **build phase** from the config class we have UVM ACTIVE_PASSIVE "enum" data type . If it is equal to UVM_ACTIVE then only will be proceed to create a master sequencer and master driver.
- In the master_agent we have a connect_phase.
- In the **connect_phase** ifuvm_active_passive_enum is_active=UVM_ACTIVE then only master driver's sequence item port will get connected to master_sequencer's sequence_item export.
- Hence creating all the elements and connection between driver and sequencer will take place in master_agent.
- Every time with the help of super.phase_name (phase) will overriding a parent class.

Master_config.

It is a class in which interface has been instantiated along with that based on user's requirement we can also add some of the static variables.

- Master_config extends from a uvm_object
- During run_time only this class will created and latter it will die.

- In master config class uvm_active_passive_enum is_active has taken which is of "enum "data type.
- In this class will be having only **function new** which use to create a class
- This enum data type value should be equal to UVM_ACTIVE value.

Master_driver

It is a class in which will be driving a input signals to the targeted design under test unit according to the protocol . In this event based delays are playing a very important part . one should be very clear about the writing protocol of SPI before to drive signals to it else will be getting undesirable result .

- Master_driver is extends from uvm_driver.
- master driver is parameterized with the master_xtn because it provides a stimulus to the signals which are used in master_xtn class.
- Uvm_component_utils use to factory registration of a class.
- In this class we have **function new, bulid_phase, connect_phase** and **run_phase**.
- In the **function new** will create a new class having a name master_driver and overrides to the parent one using super. New.
- In the **build phase**, if we won't be getting uvm_config _db from master config. then will be displaying error statement using `uvm_erro macro.
- In this class we have taken a master_driver modport.
- In **connect_phase** will be connecting master_driver modport interface to the interface taken inside a master_config class
- In **run phase** verification engineer will be writing its own logic in order drive a inputs to the targeted DUT according to the protocol.
- It is a heart of the verification plan which need so much clear understanding about the working protocol of targeted device.

Master_sequencer:

- The sequencer in UVM just acts as a bridge between sequence and driver.
- This is the only **non-virtual** class in our UVM factory.
- The sequencer is also parameterized by transaction class.
- It has **function new** which use to create a new class having a name master_sequencer.
- The sequencer takes the randomized data from sequences and passes it to the driver to which it is connected.

Master_monitor:

- It is a class which monitors all the input and output of the targeted device and send the information to the scoreboard.

master_monitor is extends from uvm_monitor.

- `uvm_component_utils is a uvm_acros use to do a factory registration of a class
- master_monitor modport virtual interface handle has also provided
- master configuration class handle has provided
- In order to communicate with the score board will be also provided with analysis_port which provides one to many connection.
- In master monitor will be having function new, build_phase, connect phase and run phase.
- In the function new will be create a new class having a name Master_monitor.
- In the build_phase if won't be able to get uvm_config_db from config_db class of master_config then error line will come.

Slave_agent_top:

It is class which enclosed the master_agent class properties as well as methods.

- Slave_agent_top is extended from uvm_envireonment.
- With the `uvm_component_utils the factory registration of slave_agent_top has been done.
- uvm_component_utils is a macros in UVM which is used for factory registration.
- With the function new will be able create a class having a name slave_agent_top and it is overridden to the parent class by super. New function.
- In the build phase of slave_agent_top will create slave_agent , for that will allocate a handle of slave_agent.
- in the run_phase will be printing a topology with uvm_top.print_topology() macro name.

It is a class which enclosed the configuration class, driver class, sequencer class and monitor class properties as well as methods.

- slave_agent is extended from uvm_agent .
- Factory registration of this class will be done using UVM macros `uvm_component_utils by passing slave_agent as an argument.
- Function new is use to create a class having a name called slave_agent which is gets overridden on the parent class.
- In the build phase of slave agent will get the configuration data from uvm_config_db winch is a macro stores the information about config class , Hence if will get the config_db then will create a slave_mmonitor else will be displaying an error massage using macro `uvm_error.
- In the build phase from the config class we have UVM ACTIVE_PASSIVE "enum" data type . If it is equal to

UVM_ACTIVE then only will be proceed to create a slave_sequencer and slave_driver.

- In the slave_agent we have a connect_phase.
- In the connect_phase if uvm_active_passive_enum is_active=UVM_ACTIVE then only slave_driver's sequence item port will get connected to slave_sequencer's sequence_item export.
- Hence creating all the elements and connection between driver and sequencer will take place in slave_agent.
- Every time with the help of super.phase_name(phase) will overriding a parent class .

slave configuration

It is a class in which interface has been instantiated along with that based on user's requirement we can also add some of the static variables.

- Slave_config extends from a uvm_object
- During run_time only this class will created and latter it will die.
- In slave config class uvm_active_passive_enum is_active has taken which is of "enum "data type.
- In this class will be having only function new which use to create a class
- This enum data type value should be equal to UVM_ACTIVE value.

Slave_driver:

It is a class in which will be driving a input signals to the targeted design under test unite according to the protocol. In this event based delays are playing a very important part . one should be very clear about the writing protocol of SPI before to drive signals to it else will be getting undesirable result .

- Slave driver is extends from uvm_driver.
- Slave driver is parameterized with the slave_xtn because it provides a stimulus to the signals which are used in slave_xtn class.
- Uvm_component_utils use to factory registration of a class.
- In this class we have function_new, bulid_phase , connect_phase and run_phase.
- In the **function new** will create a new class having a name slave_driver and overrides to the parent one using super. New.
- In the build_phase, If we won't be getting uvm_config _db from slave config then will be displaying error statement using `uvm_erro macro.
- In this class we have taken a slave_driver modport.
- In connect_phase will be connecting slave_driver modport interface to the interface taken inside a slave_config class
- In run phase verification enginner will be writing its own logic in order drive a inputs to the targeted DUT according to the protocol.
- It is a heart of the verification plan which need so much clear understanding about the working protocol of targeted device.

Slave_sequencer

- The sequencer in UVM just acts as a bridge between sequence and driver.
- This is the only non-virtual class in our UVM factory.
- The sequencer is also parameterized by transaction class.
- It has function new which use to create a new class having a name slave_sequencer.
- The sequencer takes the randomized data from sequences and passes it to the driver to which it is connected.

Slave_monitor

It is a class which monitors all the input and output of the targeted device and send the information to the scoreboard.

- Slave monitor is extends from uvm_monitor.
- uvm_component_utils is a uvm_acros use to do a factory registration of a class
- slave_monitor modport virtual interface handle has also provided
- slave configuration class handle has provided
- In order to communicate with the score board will be also provided with analysis_port which provides one to many connection.
- In slave monitor will be having function new , build_phase , connect phase and run_phase.
- In the function new will be create a new class having a name Slave_monitor.
- In the build_phase if won't be able to get uvm_config_db from config_db class of slave_config then error line will come.
- In the connect_phase connection of virtual interface to the interface which is defined in slave_configuration class.
- In the run_phase will be sending all the data to the new created transaction class from the interface based on protocol.

At last will be having a report_phase in which number of transaction for monitor will be counted.

SCOREBOARD

It is use to compare a result from master and slave side.

- scoreboard is extended from uvm_scoreboard
- It has provided with master_xtn and slave_xtn analysis fifo handles.

- It has master and slave transaction handles
- It has a handles for coverage also.
- Scoreboard has a cover group which includes a bins for every signals hence will be having an information about that how many bits from a particular signal has been triggered with the written verification plan.
- In this plan will be having two cover group. One is for master and another one is for slave.
- In the function new will be creates a master and slave cover group along with that master and slave analysis fifo handles.
- In the build_phase will be creating master_data and slave_data.
- Inside a run_phase within fork_join will be getting data from master transaction, placing handles inside analysis fifo sampling data in order to get the coverage and at last whatever response will be having from a DUT side will compare.
- It compares the data sent from one side to another i.e. from master to slave and vice-versa. This also includes cover groups which helps in coverage. It gets the data from both monitors and compares them. Also it samples the signal values which have been covered.

Virtual sequence

- Virtual sequence is also extended from uvm_sequence as normal sequences but is parameterized by uvm_sequence_item.
- It has handles of both master and slave sequences written, handles of physical sequencers and virtual sequencer.
- This class has a base sequence which will connect handles of physical sequencers to the handles of physical sequencers in virtual sequencer.
- In child sequences one has to create handles of physical sequence which has to be started and after creating we start it.

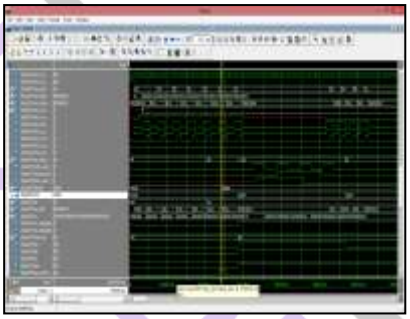
Virtual sequencer:

- Virtual sequencer is extended from uvm_sequencer and is parameterized by uvm_sequence_item.
- This is parameterized by uvm_sequence_item because this sequencer has to take both type of data i.e. master and slave.
- This sequencer is required because we cannot run virtual sequences on physical sequencers so we make a virtual sequencer and run sequences on it.
- This sequencer is pointed to parent type handle of sequencer in virtual sequence using $cast. The virtual sequencer has the handles of physical sequencers which are pointed to physical sequencers in the environment.

ENVIRONMENT

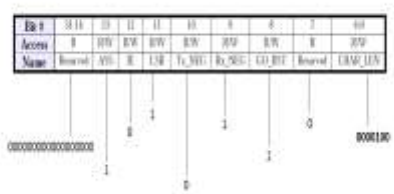
- This class is created in test and is also a most important component of our test bench. Environment creates agent-tops, scoreboard, virtual sequencer (if required) etc.
- The environment makes connection between monitors and scoreboard.
- In case if we have virtual sequencer, we have to connect physical sequencers with the handles of physical sequencers in virtual sequencer. Here we also set the local configuration db class parameters.

Output test case snippets



[CH_LEN-4 , TX_NEG-0 , RX_NEG-1 , LSB-1 , IE-1 , ASS-1 , GO_BSY-1]

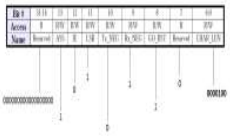
- As shown in fig here we have configured CTRL register.
- wb_dat_i for the CTRL register is 00002b04



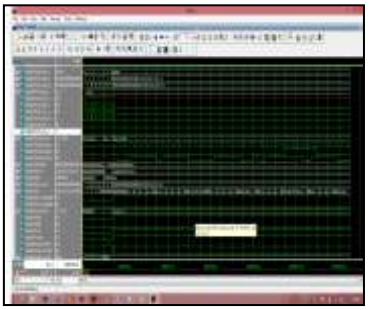
- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin
- This transfer done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it swith to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called automization

- This test case is a BUG free.

[CH_LEN-16 , TX_NEG-0 , RX_NEG-1 , LSB-1 , IE-1 , ASS-0 , GO_BSY-1]

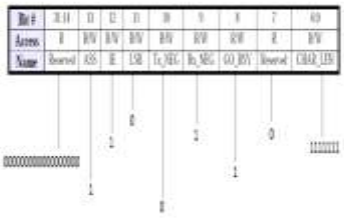


- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin
- This transfers done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called atomization
- In this test case found a BUG where at the time of receiving in the receive register last bit missed, because of int_out signal.



[CH_LEN-127 , TX_NEG-0 , RX_NEG-1 , LSB-0 , IE-1 , ASS-1 , GO_BSY-1]

- . As shown in fig here we have configured CTRL register .
- As shown in fig here we have configured CTRL register .
- wb_dat_i for the CTRL register is 00003304



- As shown in waveform character length is 127 which sent 127 bits from the selected register which can be observed from waveform on MOSI pin
- This transfer done based on the serial clock.
- as we have kept ASS=1 and IE=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then
- after it again switch to the "ff" value called atomization
- Here we can have effect of int_o high at the end of every transfer.

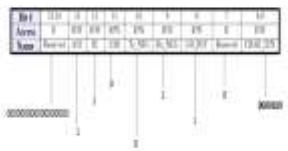
[CH_LEN-4 , TX_NEG-1 , RX_NEG-0 , LSB-0 , IE-1 , ASS-1, GO_BSY-1]

- As shown in fig here we have configured CTRL register.
- wb_dat_i for the CTRL register is 00003504

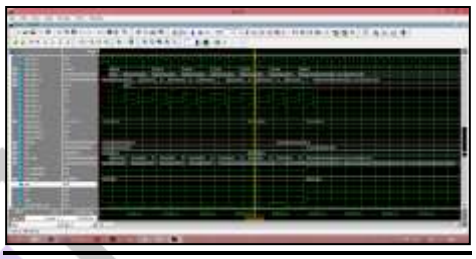
- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin
- This transfer done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called atomization
- In this test case found a BUG where at the time of receiving in the receive register last bit missed, because of int_out signal.

[CH_LEN-4 , TX_NEG-0 , RX_NEG-1 , LSB-0 , IE-1 , ASS-1 , GO_BSY-1]

- As shown in fig here we have configured CTRL register .
- wb_dat_i for the CTRL register is 00003304.

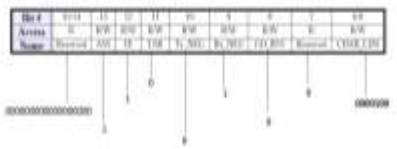


- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin
- This transfer done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called atomization
- In this test case found a BUG where at the time of receiving in the receive register last bit missed, because of int_out signal.

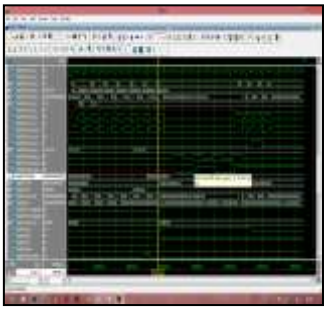


[CH_LEN-4 , TX_NEG-0 , RX_NEG-1 , LSB-0 , IE-1 , ASS-1 , GO_BSY-0]

- As shown in fig here we have configured CTRL register .
- wb_dat_i for the CTRL register is 00003204

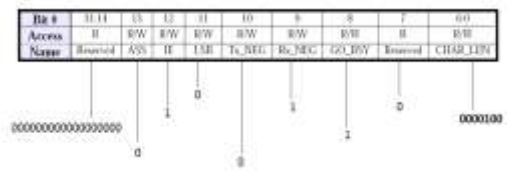


- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin
- These transfers done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called atomization
- In this case we have kept GO_BSY signal bit low hence the transfer of bits on MOSI or MISO pin will be not going to take place along with that from the output waveform we can observe that serial clock is not getting generated.



[CH_LEN-4 , TX_NEG-0 , RX_NEG-1 , LSB-0 , IE-1 , ASS-0 , GO_BSY-

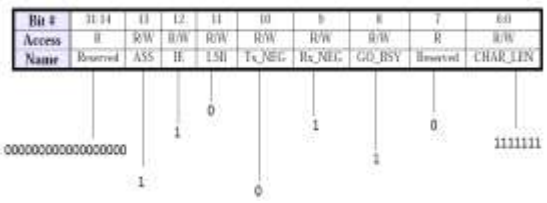
- As shown in fig here we have configured CTRL register.
- wb_dat_i for the CTRL register is 00001304



- As shown in waveform character length is 4 which sent 4 bits from the selected register which can be observed from waveform on MOSI pin.
- This transfer done based on the serial clock.
- as we have kept ASS=0 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again never switch to the "ff" value called atomization
- In this case we have kept GO_BSY signal bit low hence the transfer of bits on MOSI or MISO pin will be not going to take place along with that from the output waveform we can observe that serial clock is not getting generated.

[CH_LEN-127, TX_NEG-0 , RX_NEG-1, LSB-0 , IE-1 ,ASS-1 GO_BSY-

As shown in fig here we have configured CTRL register.



- wb_dat_i for the CTRL register is 0000337f
- As shown in waveform character length is 127which sent 127 bits from the selected register which can be observed from waveform on MOSI pin.
- This transfer done based on the serial clock.
- as we have kept ASS=1 from the waveform we can observed that ss_pad_o line initially it is "ff" but as soon as we have loaded data in slave select register it switch to "fe" value and remain on that value till 4 bit of data transfer from master core to targeted slave device has been completed and then after it again switch to the "ff" value called atomization
- In this case we have kept GO_BSY signal bit low hence the transfer of bits on MOSI or MISO pin will be not going to take place along with that from the output waveform we can observe that serial clock is not getting generated.

COVERAGE REPORT

Coverage report has been obtained by merging all the test cases. The coverage for master is 94.44% and for slave is 100%. So over all coverage is average of master and slave which is 98.61%. Included assertions and got the weighted average of coverage which is 98.61%.[1][4][10]

Weighted Average:				98.61%
Coverage Type	Bins	Hits	Misses	Coverage (%)
Covergroup	23	21	2	97.22%
Assertion Attempted	71	71	0	100.00%
Assertion Failures	71	0	-	0.00%
Assertion Successes	71	71	0	100.00%

[Weighted avg]

[master coverage]

Covergroup type slv coverage				100.00%
Coverpoints / Bins	At Least	Hits	Goal	Coverage
Coverpoint MISO_PAD_1			100.00%	100.00%

[slave coverage]

Mrs.SHYAMALA S C received the B.E. degree in Electronics and communication Engineering from JNNCE, Shivamogga in the year 2002 and M.Tech in VLSI and Embedded from SJCE, Mysore, Karnataka in 2013. She is an Assistant Professor in the department of Electronics and Communication Engineering PESITM, Shivamogga. She is in teaching for 8 years. Her subjects of interest include communication engineering, control system circuits and power Electronics.

CONCLUSION

By concluding above we can say that the data transmission and reception is full duplex and its of variable length which can increase up to 128 bits And data transfer can be initiated through MSB or LSB bit, Rx and Tx on both rising or falling edge of serial clock independently, 8 slave select line are available ,fully static synchronize design with one clock domain, technology independent Verilog and fully synthesizable so we can say like that fully synchronize serial interface with different device like microcontroller ,DACs, ADCs and other.

REFERENCES

1.SPI Block Guide v3.06; Motorola/Freescale/NXP; 2003.

2. *"Better SPI Bus Design in 3 Steps". dorkbot pdx. Retrieved 3 September 2015.*

3. Maxim-IC application note 3947: "Daisy-Chaining SPI Devices"

4.AVR910 - In-system programming

5. SPI Adapter with support of custom serial protocols.

6. Queued Serial Module Reference Manual, Freescale Semiconductor.

7. MICROWIRE Serial Interface National Semiconductor Application Note AN-452

8.MICROWIRE/PLUS Serial Interface for COP800 FamilyNational Semiconductor Application Note AN-579

9.Serial Peripheral Interface (SPI) Flash Memory Backgrounder

1. *"Intel® 100 Series Chipset Family PCH Datasheet, Vol. 1" (PDF). Retrieved April 15, 2015.*